Application of 2D Packing Algorithms to the Woodwork Industry

Tiago Ribeiro - a40309

Thesis presented to Escola Superior de Tecnologia e de Gestão de Bragança to obtain the master's degree in Industrial Engineering - Mechanical.

Orientation by:

Prof. Ana Isabel Pereira

This thesis does not include any critics or suggestions done by the juri.

Bragança

2022-2023

Application of 2D Packing Algorithms to the Woodwork Industry

Tiago Ribeiro - a40309

Thesis presented to Escola Superior de Tecnologia e de Gestão de Bragança to obtain the master's degree in Industrial Engineering - Mechanical.

Orientation by:

Prof. Ana Isabel Pereira

This thesis does not include any critics or suggestions done by the juri.

Bragança

2022-2023

Resumo

Esta pesquisa investiga a aplicação de metodologias computacionais na indústria madeireira, com foco no Problema do Corte de Material (PCE) com duas iterações: guilhotinável e não guilhotinável. O estudo aplica um algoritmo evolucionário baseado no Non-dominated Sorting Genetic Algorithm II (NSGA-II) adaptado às complexidades do problema para otimizar o processo de corte. A metodologia tem como objetivo melhorar a eficiência da utilização de material em tarefas de trabalho em madeira, empregando este algoritmo utilizando sobras de peças ao invés de uma nova placa. O relatório fornece dados empíricos e métricas de desempenho do algoritmo, demonstrando a sua eficácia na redução do desperdício e na otimização do trabalho na indústria. Esta abordagem melhora a eficiência operacional e sublinha os benefícios ambientais da utilização mais sustentável dos recursos de madeira, exemplificando o potencial da integração de técnicas computacionais em indústrias tradicionais para atingir este objetivo.

Keywords: Otimização, Otimização multi-objetivo, Problemas de empacotamento 2D, Problemas de corte de material 2D

Abstract

This research investigates the application of computational methodologies in the woodworking industry, focusing on the Cutting Stock Problem (CSP) with two iterations: guillotinable and non-guillotinable iterations. The study applies an Evolutionary Algorithm (EA) based on Non-dominated Sorting Genetic Algorithm II (NSGA-II) customized to fit the intricacies of the problem to optimize the cutting process. The methodology aims to enhance material usage efficiency in woodworking tasks by employing this algorithm using leftover parts instead of a new board. The report provides empirical data and performance metrics of the algorithm, demonstrating its effectiveness in reducing waste and optimizing labor in the industry. This approach improves operational efficiency and underscores the environmental benefits of using timber resources more sustainably, exemplifying the potential of integrating computational techniques in traditional industries to achieve this objective.

Keywords: Optimization, Multi-objective optimization, 2D Packing problems, 2D Cutting Stock Problems

Contents

List of Tables

List of Figures

Acronyms

2DPP 2D Packing Problem. 1, 3, 7

CSP Cutting Stock Problem. vii, 1–3, 7–10, 29, 37

DEAP Distributed Evolutionary Algorithms in Python. 16, 17, 21–23

EA Evolutionary Algorithm. vii, 2, 16–21, 25, 27, 29, 33, 34, 37, 39, 46, 47, 49, 50, 52

GEOS Geometry Engine - Open Source. 17

GIS geographic information systems. 17

JTS Java Topology Suite. 17

NSGA-II Non-dominated Sorting Genetic Algorithm II. vii, 2, 21, 29

SCOOP Scalable Concurrent Operations in Python. 16

Chapter 1

Introduction

The Wood Work 4.0 (WW4.0) project aims to develop new approaches to how furniture production is carried out, mainly in Small and Medium Enterprises (SMEs). As a sector that has been modernized through the introduction of new machines and new processes, how some of the internal processes are still managed is still at a very archaic stage and impacts the overall operation of the system. Thus, the WW4.0 project aims to develop new approaches that allow the total digitization of the internal processes of the furniture production chain in such a way that they are integrated into a global approach.

This work aligns with the objectives of the WW4.0 project by addressing a crucial yet underexplored aspect in the woodworking industry: the efficient reuse of leftover boards in cutting stock algorithms. This study addresses this by developing a cutting stock algorithm that solves the CSP using leftover boards and new boards. This approach improves the efficiency of wood utilization in cutting operations and promotes sustainability by prioritizing leftover materials before resorting to new ones.

2D Packing Problem (2DPP) and CSP are fundamental in computational geometry and operations research. These problems involve the strategic arrangement of various objects within one or more containers to maximize space utilization while adhering to the objects' physical constraints. In the woodworking industry, this translates to the optimal placement of different cutting patterns on wood boards to minimize waste and maximize material usage. This aspect of computational optimization is crucial in industries where raw materials are costly and their conservation is environmentally significant.

EA play an important role in this study. These algorithms, inspired by biological evolution, apply selection, mutation, and crossover mechanisms to evolve solutions to optimization problems iteratively. The research uses a custom-tailored EA based on NSGA-II, particularly suited for the complexities of the CSP. This approach is chosen for its robustness in handling multi-objective optimization tasks and its effectiveness in navigating the vast solution spaces in CSP.

The structure of this thesis is as follows: Chapter 1 presents the introduction; Chapter 2 provides a comprehensive review of pertinent literature; Chapter 3 defines the problem, distinguishing between the guillotinable and non-guillotinable cutting stock challenges; Chapter 4 elaborates on the methods and techniques utilized; Chapter 5 delineates the developed solution; Chapter 6 discusses the results; and Chapter 7 offers conclusions and outlines directions for future research.

Chapter 2

Literature Review

A comprehensive literature review is essential to understand the current state and future possibilities in an evolved woodworking industry, where traditional craftsmanship blends with modern technology. This review examines the intersection of advanced computer science techniques, mainly 2DPP and CSP, with the woodworking sector. Despite its evolution, the industry still faces material efficiency and sustainability challenges, areas ripe for further computational innovation. This review seeks to see how computational methods have been and can be further integrated into woodworking to optimize material usage and increase productivity.

Muhammed Beyaz et al. [2] delve into the intricacies of the offline 2D bin-packing problem (2DBPP), a well-recognized NP-hard combinatorial optimization challenge. In this problem, objects of varying width and length dimensions must be efficiently packed into a minimal number of 2D bins. Although several heuristic methods were proposed in the past, exact solutions for larger problem instances have yet to be discovered. Traditional methods such as next-fit, first-fit, best-fit, unified tabu search, and genetic and memetic algorithms were successfully applied. In this study, the authors introduce a novel set of hyper-heuristic algorithms that ingeniously select and combine the best features of state-of-the-art heuristics and local search techniques. The primary objective is to minimize the number of 2D bins used. A standout feature of these proposed algorithms is the introduction of new crossover and mutation operators designed explicitly for heuristic selection. The robustness and efficiency of the algorithms are demonstrated through extensive experiments on benchmark problem instances for offline 2DBPP. The results are promising and showcase the algorithm's ability to consistently achieve solutions close to optimal.

Brian Brubach et al. [3] focus is directed toward exploring column-sparse packing problems, a subset of combinatorial optimization problems that have garnered significant attention due to their intricate nature and wide-ranging applications. These problems involve the challenge of efficiently packing items into containers, considering the sparsity of the packing matrix columns. The primary objective is to achieve an optimal or near-optimal packing solution while adhering to the constraints imposed by the sparsity. Brubach and his team introduce two groundbreaking ideas in this domain: attenuation and multiple-chance algorithms. These concepts are innovative strategies to derive improved approximation algorithms for column-sparse packing problems. The attenuation approach involves adjusting the packing constraints to achieve a more balanced and efficient solution. On the other hand, multiple-chance algorithms provide a probabilistic framework, offering multiple opportunities to achieve an optimal packing configuration. Through rigorous experimentation and analysis, the authors demonstrate the efficacy of these methods in obtaining solutions that closely approximate the optimal.

Balcar et al. [1] delve into a unique class of 2D stock cutting problems, precisely the semi-guillotinable problems. This classification is paramount when devising optimal cutting plans for circular saws, a standard tool in various manufacturing processes. The authors introduce a new algorithm specifically tailored for both guillotinable and non-guillotinable 2D cutting stock problems. Through their research, Balcar and his team aim to bridge the gap between traditional cutting methods and the evolving demands of modern manufacturing. Their approach offers a fresh perspective on optimizing material usage, ensuring minimal waste, and maximizing efficiency in the cutting process.

Evtimov and Fidanova [6] [5], address the challenges posed by the 2D cutting stock problem, especially when the items are irregular polygons. Recognizing the inherent complexity and computational challenges of the problem, the authors propose a novel stochastic algorithm as a solution. This algorithm handles the intricacies of cutting irregularly shaped items from larger stock material, ensuring optimal utilization and minimal wastage. The paper underscores the importance of developing advanced heuristic manufacturing and material optimization methods. The approach of Evtimov and Fidanova stands out for its ability to provide efficient solutions to real-world cutting problems, emphasizing the potential of heuristic algorithms in addressing complex industrial challenges.

Cintra et al. [4] delve into the intricacies of cutting stock problems and their approximability. The authors shed light on the similarities and differences between cutting stock problems and bin-packing problems, emphasizing the variability of input items as a distinguishing factor. Their research demonstrates that the one-dimensional cutting stock problem is as challenging to approximate as the bin-packing problem. Furthermore, they establish that the two-dimensional cutting stock problem shares the same level of approximability difficulty as its two-dimensional bin packing counterpart. Cintra and his team contribute significantly to the theoretical understanding of cutting stock problems, providing valuable insights into this domain's challenges and potential solutions.

Considering the state-of-the-art, it is possible to observe that computational methodologies, particularly heuristic and hyper heuristic algorithms, have progressively been employed to address the intricate challenges of 2D packing and cutting in traditional industries. From the introduction of novel operators in bin-packing by Muhammed Beyaz et al. to the specialized algorithms for guillotinable problems by Balcar et al., and the unique stochastic approach by Evtimov and Fidanova for irregular polygons, there is a clear trajectory toward optimizing material utilization and minimizing wastage. Furthermore, the deep theoretical insights provided by Cintra et al. elucidate the complexities and parallels between cutting stock and bin-packing problems. These advances underscore the potential of computational techniques to revolutionize the woodworking industry and establish a solid foundation for the methodologies and approaches explored in this thesis.

Chapter 3

Problem Definition

This chapter provides a detailed exploration of the CSP, also known as the 2DPP. Emphasis is placed on the efficient utilization of previously used materials, highlighting the distinction between guillotinable and non-guillotinable problem types. The mathematical framework, including parameters, decision variables, objective functions, and constraints, are systematically presented to offer a comprehensive understanding of the problem's intricacies.

3.1 2D Cutting Stock Problem

This project addresses the CSP, the primary objective is to optimize the use of previously used stock material sourced from a database. By doing so, we aim to reuse leftovers, thus recycling materials that might otherwise be discarded, promoting sustainable practices, and maximizing the value derived from each material piece.

In the CSP there are two different sub-problems, guillotinable and non-guillotinable. For the successful completion of this project, it is imperative to devise solutions for both the guillotinable and non-guillotinable problems, ensuring maximum material utilization in each scenario.

3.1.1 Guillotinable Cutting Stock Problem

In the guillotinable CSP, the cuts are made sequentially, where each cut extends from one edge of the material to the opposite edge without interruptions. This problem ensures that the material is divided into distinct rectangular pieces needed for some of the machinery used in woodworking. The advantage of guillotinable cuts is their simplicity and efficiency; however, they may only sometimes yield the most optimal use of material, especially when the desired pieces have irregular shapes.

3.1.2 Non-Guillotinable Cutting Stock Problem

The non-guillotinable CSP allows for cuts that do not necessarily extend from one edge to another, providing greater flexibility in the cutting patterns and can accommodate more complex or irregular shapes. While this problem can potentially result in a more optimized use of material, it might be more challenging to implement, especially when using automated machinery.

3.2 Mathematical Formulation

Considering the problem of packing $P_1, P_2, ..., P_n$ rectangular pieces, where P_i is defined by its vertices $(v_1^i, v_2^i, v_3^i, v_4^i)$, as represented in Figure 3.1:

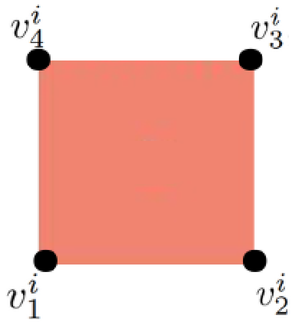


Figure 3.1: Vertices Representation

Let $x = [y_1, y_2, ..., y_n, Y_1, Y_2, ..., Y_n]$, where $y_i \in \{P_1, P_2, ..., P_n\}$ and $Y_i \in \{0, 1\}$ represent a possible solution to the proposed problem. For each element x, it is necessary to calculate the length used in the cut, l, the height used in the cut, h, and the vertical misalignment of all pieces, s.

3.2.1 Parameters

The CSP involves specific parameters that are set at the beginning and do not change during the solution-finding process, namely:

- The **wood board** is represented as a collection of points, each point corresponding to a vertex of its polygonal shape, as illustrated in Figure 3.3. Rectangular boards can alternatively be defined by specifying their length (L) and height(H), as defined in Figure 3.2.

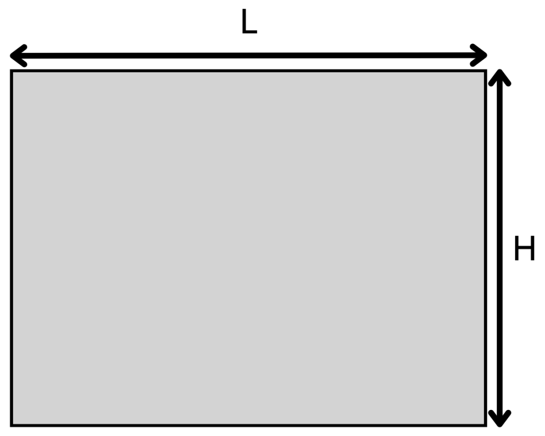


Figure 3.2: Wood board $H \times L$

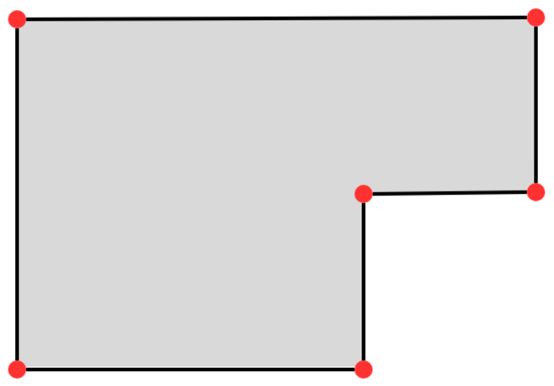


Figure 3.3: Wood board point representation

- The **pieces** are represented as rectangles; for more intricate shapes, they should be represented by their bounding rectangle. The cuts are enumerated in a list, each entry detailing the length (l_i) and height (h_i) of the respective cut, as illustrated in Figure 3.4. If multiple cuts of the same type are required, they must be repeated in the list.

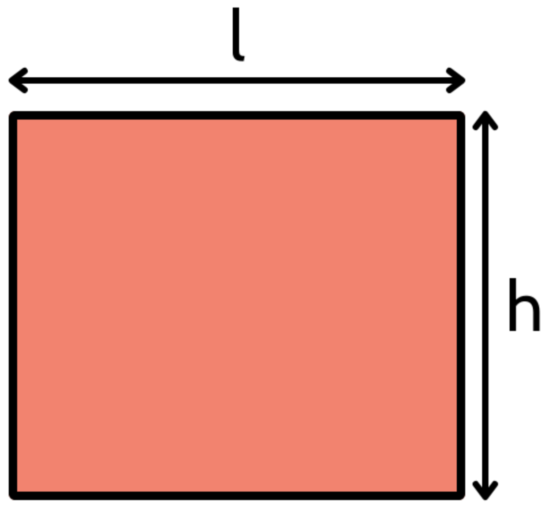


Figure 3.4: Piece representation

3.2.2 Decision Variables

In the CSP, two sets of variables are defined at the beginning, namely:

- The **placement sequence** determines how the cuts are placed on the board. Modifying this sequence results in various solutions associated with different fitness values. The efficacy of this procedure stems from the heuristic prioritizing placing a cut closer to the origin. When multiple placement points are equidistant from the origin, the location with the lowest height is selected. The placement sequence is represented by $y_i \in \{P_1, P_2, ..., P_n\}$.

- The **rotation state** is a list that mirrors the length of the placement sequence. It comprises binary values: 0 or 1. A value of 0 denotes a rotated state, while 1 signifies a non-rotated state. Notably, the rotation is consistently set at 90º. The rotation state is represented by $Y_i \in \{0, 1\}$.

3.2.3 Objective function

Although the parameters and decision variables are consistent in both guillotinable and non-guillotinable problems, their objective functions display substantial distinctions.

Guillotinable

For the guillotinable problem, the objectives are defined as follows:

- $f_1(x) = h$, representing the total height used in the solution.
- $f_2(x) = s$, where $s = \sum_{i=1}^{n} g(x_i)$, where $g(x_i) = \begin{cases} h_{i-1} - h_i & \text{if } h_i \leq h_{i-1} \\ 2(h_i - h_{i-1}) & \text{if } h_i > h_{i-1} \end{cases}$, this represents the sum of the vertical misalignment of piece i in relation to the previous piece.

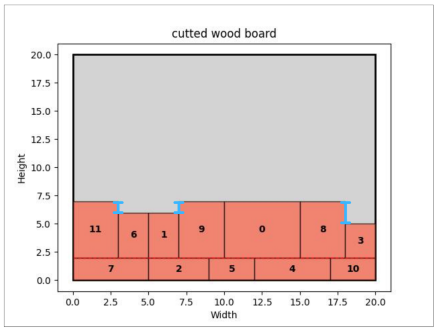


Figure 3.5: Vertical misalignment visualization (in blue)

- $f_3(x) = A_{env} - \sum_{i=1}^{n} A_i$, where A_{env} represents the envelope area, which is the length multiplied by the height occupied by the pieces ($l \times h$), as represented in Figure 3.6, and A_i represents the area of the piece x_i.

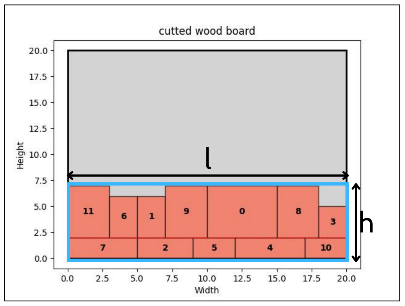


Figure 3.6: Envelope visualization

Non-guillotinable

- $k_1(x) = A_{env} - \sum_{i=1}^{n} A_i$, where A_{env} represents the envelope area and A_i represents the area of the piece x_i.
- $k_2(x) = \frac{l+h}{2}$, indicative of the average utilization of the board height and length by the pieces.

3.2.4 Constraints

The constraints set for this problem are broad-based, ensuring that any derived solution is feasible in practical applications:

- **Completion of Cuts:** All cuts must be executed for a solution to be deemed successful.
- **Non-overlapping Cuts:** There must be no overlap between any cuts. Although cuts can share boundaries, overlapping is strictly prohibited.
- **Boundary Constraint:** All cut placements must be entirely contained within the confines of the wood board.

3.3 Problem Formalization

As discussed earlier, the initial step in addressing the problem involves defining the parameters. Given these parameters, the solution aims to identify decision variables that most effectively minimize the values of the objective function. Consequently, the anticipated output is structured as follows:

- **Best Individual:** This represents the optimal placement sequence derived from the process.
- **Rotated Indices:** Details of the rotation state are captured as rotation indices.

- **Fitness Measure:** The values of the objective functions quantify the efficacy of the solution.
- **Execution Time:** The time taken for the computation is provided, measured in seconds.

Chapter 4

Methods and Techniques

The choice of software tools and libraries for a research project significantly influences its outcomes, ensuring accuracy, repeatability, and efficiency by opting for robust and appropriate software. This chapter provides a review of the software and libraries used in this research, outlining the rationale behind each choice, and elucidating how they contribute to the project objectives.

4.1 Python and Software

Python, the language chosen for this research, has gained immense popularity in the scientific community due to its simplicity, readability, and vast ecosystem of specialized libraries. Python's dynamic and interpreted nature facilitates rapid prototyping and iterative development, making it an ideal choice for research endeavors where flexibility and adaptability are paramount [13].

In the context of this research, Python was chosen for the following reasons:

- Strong Community Support: The expansive Python community regularly contributes to its rich ecosystem, ensuring up-to-date tools and solutions for many challenges.
- Interdisciplinary Integration: Python's wide-ranging libraries cater to diverse domains, from data analytics with pandas to machine learning with TensorFlow and

SciKit-learn. This breadth enables holistic research that spans several scientific disciplines.

- Ease of Use: Python's syntax is designed for clarity, making it accessible for programming veterans and newcomers. This quality ensures that the research focuses on the core objectives rather than getting bogged down by programming complexities [13].

4.1.1 Software Libraries

Each research challenge requires specialized tools. This research harnesses state-of-the-art solutions tailored for specific tasks by leveraging Python's extensive library ecosystem. In the following sections, each library's functionalities, relevance to the research, and justifications for its inclusion will be meticulously detailed.

DEAP

Distributed Evolutionary Algorithms in Python (DEAP) is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It seeks to make algorithms explicit and data structures transparent. It works in perfect harmony with parallelization mechanism such as multiprocessing and Scalable Concurrent Operations in Python (SCOOP) [7].

A major advantage of using the DEAP library is its comprehensive suite of methods for implementing an EA, which offers both flexibility and robustness. This research uses the following methods:

- **Creator**: Facilitates the creation of individuals, the population, and the establishment of the fitness function.

- **Toolbox**: Serves to initialize the functions defined in the *"creator"* method and also sets up the mechanisms for crossover and mutation.

- **Algorithms**: Execute the specified algorithm to guide the evolutionary process.

Through the capabilities listed above, the DEAP library streamlines the process of crafting and refining EA. The subsequent chapters delve deeper, providing intricate details on the diverse components integral to the functioning of the EA.

Shapely

Shapely is a Python package for set-theoretic analysis and manipulation of planar features using functions from the well-known and widely deployed Geometry Engine - Open Source (GEOS) library. GEOS, a port of Java Topology Suite (JTS), is the geometry engine of the PostGIS spatial extension for the PostgreSQL RDBMS. The designs of JTS and GEOS are largely guided by the Open Geospatial Consortium Simple Features Access Specification [10] and Shapely adheres primarily to the same set of standard classes and operations. Shapely is thereby deeply rooted in the conventions of the geographic information systems (GIS) world, but aspires to be equally useful to programmers working on non-conventional problems [8].

Shapely played a pivotal role in the research, facilitating geometric operations on geometric bodies. Specifically, with Shapely, we were able to:

- Construct polygons using the set of points from each geometry.
- Determine overlaps and ensure that no geometries extend out of bounds.
- Compute areas and measure distances.

Subsequent chapters will delve deeper into these functionalities, providing a detailed step-by-step account of how each capability was effectively utilized.

Matplotlib

Matplotlib is a 2D graphics package used for application development, interactive scripting, and publication quality image generation across user interfaces and operating systems [11].

In this research, matplotlib served primarily as a visualization tool, allowing a clearer understanding of the solutions obtained. Leveraging the library's 2D plotting capabilities, we combined it with Shapely's polygon geometries to depict the optimal solution.

NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, and much more [9].

NumPy was the primary tool for managing arrays and data structures in this project. These structures, crafted by the researcher, were integral to various libraries used throughout the research.

4.2 Evolutionary Algorithm Overview

EA form a family of optimization and search algorithms inspired by the principles of natural selection and genetics. They operate on a population of potential solutions to a given problem, evolving this population over time through the application of bio-inspired operators: selection, mutation, crossover, and occasionally migration [12].

The fundamental idea behind EA is the iterative process of generating new candidate solutions based on the quality (or fitness) of existing solutions. This mirrors the natural evolutionary principle, where individuals that are better adapted to their environment have a higher chance of reproducing and passing on their genetic material to the next generation.

Key components and processes in EA include:

- **Representation of solutions**: Solutions in EA are typically encoded as chromosomes. These can take various forms, such as binary strings, real-valued vectors, or more complex data structures, depending on the domain of the problem.

- **Initialization**: An initial population of potential solutions is generated, usually randomly.

- **Evaluation**: A fitness function assesses the quality or suitability of each solution in the current population.

- **Selection**: Solutions are chosen to form a mating pool. The likelihood of a solution being selected is often (but not always) proportional to its fitness, reflecting the principle of survival of the fittest.

- **Crossover (Recombination)**: Pairs of solutions from the mating pool are combined, creating offspring that inherit features from both parents.

- **Mutation**: This introduces small, random changes in solutions, ensuring genetic diversity within the population and aiding exploration of the solution space.

- **Replacement**: New solutions replace old ones in the population, and the algorithm returns to the evaluation phase.

- **Termination**: The algorithm stops when a stopping criterion is met, which could be a maximum number of generations, a satisfactory fitness level being achieved, or other criteria.

EA are highly adaptable and have been extended and modified to form various specialized algorithms, such as Genetic Algorithms, Evolutionary Strategies, and Differential Evolution, among others. Their inherent flexibility allows them to be applied to a wide range of optimization problems, from simple function optimization to more complex real-world scenarios. Figure 4.1 presents the flowchart of an EA.

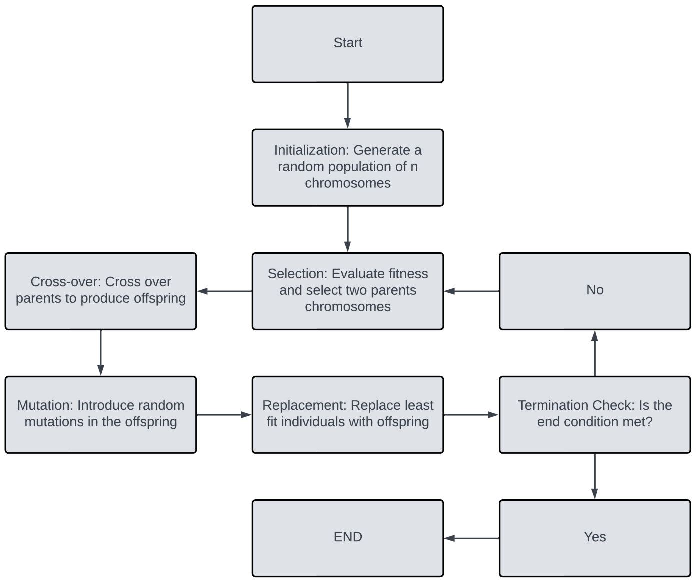

Start
Initialization: Generate a random population of n chromosomes
Cross-over: Cross over parents to produce offspring
Selection: Evaluate fitness and select two parents chromosomes
No
Mutation: Introduce random mutations in the offspring
Replacement: Replace least fit individuals with offspring
Termination Check: Is the end condition met?
END
Yes

Figure 4.1: Evolutionary Algorithm process

4.3 Evolutionary Algorithm Procedures

An EA can have a variety of procedures, including representation, initialization, selection, crossover, mutation, and algorithm. The following subsections will explain each of these in more detail.

4.3.1 Representation

In the context of our EA design, representation refers to the way solutions or chromosomes are represented within the problem framework. For this specific problem, we use two different types of variables, $x = [y_i, Y_i]$. The first variable, y_i is a sequence of single integers from 1 to the number of desired polygonal cuts, $y_i \in \{P_1, P_2, ..., P_n\}$. This sequence ensures that each polygon is identified and avoids repetition. The second variable, Y_i is a binary sequence composed of 0's and 1's, showing the rotation status of each polygon or cut, $Y_i \in \{0, 1\}$. Here, a '0' means a non-rotated state, while a '1' means a rotational state.

4.3.2 Initialization

The initial population is constructed using a dedicated function that generates the "gen 0" chromosomes. This function is invoked repeatedly based on the specified number of individuals within the population, ensuring that each individual is uniquely initialized.

By invoking the `generate_individual()` function, a chromosome is obtained consisting of two parts: a unique sequence of indices representing the cuts and a binary sequence indicating the rotation state of each cut.

4.3.3 Selection

The selection process in an EA determines which solutions of the current population will produce the next generation, favoring better solutions and allowing them to pass their genes on to the next generation.

The DEAP library has an extensive list of selection operators already programmed, as follows:

- `selTournament()`
- `selRoulette()`
- `selNSGA2()`
- `selNSGA3()`
- `selSPEA2()`
- `selRandom()`
- `selBest()`
- `selWorst()`

Of all the options tested, the selection operator chosen was `selNSGA2()` because it best fits the problem. The `selNSGA2()` applies the NSGA-II selection operator on individuals, which handles the selection of solutions based on their dominance relationship in the objective space, maintaining diversity while ensuring that non-dominated solutions are prioritized [7].

4.3.4 Crossover

Given the multi-variable nature of the problem, which involves two distinct types of variables, the crossover operator is tailored for each variable type. The DEAP library provides a plethora of options for the crossover operator, as listed below.

- `cxOnePoint()`
- `cxTwoPoint()`
- `cxUniform()`
- `cxPartialyMatched()`
- `cxUniformPartialyMatched()`
- `cxOrdered()`
- `cxBlend()`

After thorough research and empirical testing of the available operators, the chosen for the sequence of integers is the `cxUniformPartialyMatched()` (Figure 4.2), because it expects sequence individuals of indices [7] ensuring that the offspring maintains the permutation property, meaning that there are no repeated elements and all elements are taken into account.

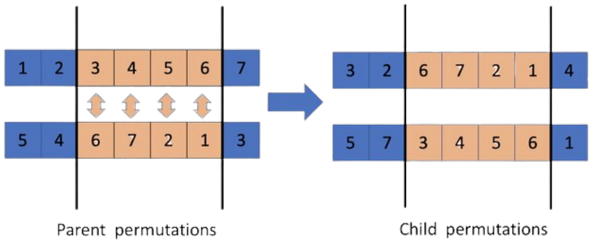


Figure 4.2: Uniform Partially Matched Crossover

For the binary sequence of 0's and 1's, the crossover operator chosen was `cxTwoPoint()` (Figure 4.3), which executes a two-point crossover on the input sequence individuals, modifying them in place and both keeping their original length [7].

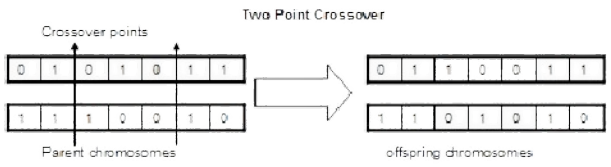


Figure 4.3: Two-Point Crossover

4.3.5 Mutation

As the crossover operator, the mutation is tailored for each type of variable, where the DEAP library offers a multitude of options, as listed below.

- `mutGaussian()`
- `mutShuffleIndexes()`
- `mutFlipBit()`
- `mutPolynomialBounded()`
- `mutUniformInt()`
- `mutESLogNormal()`

After testing various operators, the chosen for the sequence of integers was the `mutShuffleIndexes()` (Figure 4.4) which shuffles the attributes of the input individual and returns the mutant. The individual is expected to be a sequence. The `indpb` argument is the probability of each attribute to be moved. Usually, this mutation is applied to a vector of indices [7].

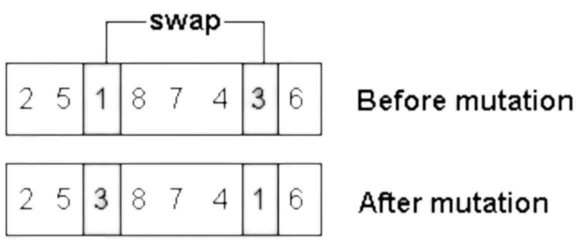


Figure 4.4: Shuffled Index Mutation

The mutation operator for the binary sequence was `mutFlipBit()` (Figure 4.5) which changes the value of the attributes of the input individual and returns the mutant. This mutation is usually applied to Boolean individuals.

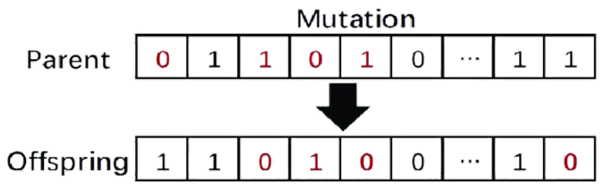


Figure 4.5: FlipBit Mutation

4.3.6 Algorithm

The algorithm used in the project was the $\mu+\lambda$ evolutionary algorithm. The name refers to the way parents and offspring are selected for the next generation, where μ represents the number of parents and λ represents the number of offspring. The algorithm works as follows:

1. **Initialization**: Start with an initial population of μ individuals.

2. **Offspring Production**: From this population, λ offspring are produced, typically through procedures such as crossover, mutation, and reproduction.

3. **Selection**: The offspring are then evaluated, and the next generation population is selected from both the offspring and the population based on their fitness.

The essential characteristic of the strategy $\mu+\lambda$ is that parents can survive to the next generation if they are better than their offspring, which contrasts with the μ, λ strategy, where the next generation is selected entirely from the offspring, and no parent is allowed to survive unless it is reproduced as an offspring [7].

The $\mu + \lambda$ strategy has various advantages, such as:

- it allows good solutions to persist over generations, which can be beneficial if the offspring does not outperform their parents;

- combining parents and offspring in the selection pool can increase genetic diversity, potentially leading to better solution space exploration.

The algorithm takes a population and evolves it using the `varOr()` function, returning the optimized population and a logbook with the evolution statistics [7].

The `varOr()` function is part of an EA applying only the variation part (crossover, mutation, or reproduction). Individuals are cloned, so the returned population is independent of the input population [7].

In the case of a crossover, two individuals are selected at random from the parental population; those individuals are cloned and then mated using the `toolbox.mate()` procedure. Only the first child is appended to the offspring population; the second child is discarded.

In the case of a mutation, one individual is selected at random, cloned, and then mutated using the `toolbox.mutate()` procedure. The resulting mutant is added to the offspring population.

In the case of a reproduction, one individual is selected at random, cloned, and appended to the offspring population.

This variation is named `Or` because an offspring will never result from both operations, crossover and mutation. The sum of both probabilities shall be in the interval $[0, 1]$.

4.3.7 Termination criteria

The termination criteria for this problem are based on the number of consecutive generations without any improvement. Specifically, the algorithm monitors the best solution of each generation and compares it with the best solution of the previous generation. The algorithm terminates if the fitness values remain unchanged for a predefined number of generations.

4.4 Implementation Details

For the geometric operations integral to this project, the Shapely library was employed. Shapely is a Python package that provides an extensive set of methods for manipulating and analyzing planar geometric objects. Several reasons underpin the decision to utilize Shapely:

- Extensive Geometric Operations: Shapely offers a comprehensive toolkit for geometric operations, ensuring that a wide range of geometric tasks can be executed efficiently and accurately.
- Ease of Integration: Shapely's Pythonic interface integrates easily with other Python libraries and tools. Its compatibility with famous data structures makes it a versatile choice for projects that require geometric computations.
- Reliability: Built on the robust GEOS library, Shapely ensures that geometric operations are extensive and reliable, providing consistent and accurate results across various use cases.

The methods employed in this project are delineated below.

- `shapely.Point` - A geometry type that represents a single coordinate with x, y and possibly z values. A point is a zero-dimensional feature and has zero length and zero area.
- `shapely.Polygon` - A geometry type representing an area that is enclosed by a linear ring. A polygon is a two-dimensional feature and has a non-zero area. It may have one or more negative-space "holes" which are also bounded by linear rings.
- `shapely.bounds` - Computes the bounds (extent) of a geometry. For each geometry, these 4 numbers are returned: $(min_x, min_y, max_x, max_y)$.
- `shapely.contains` - Returns True if geometry B is completely inside geometry A.
- `shapely.touches` - Returns True if the only points shared between A and B are on the boundary of A and B.
- `shapely.union` - Merges geometries into one.

- `shapely.envelope` - Computes the minimum bounding box that encloses an input geometry.

Matplotlib was employed within this project's scope for its superior plotting capabilities. This renowned Python library facilitated data visualization and results throughout the project. Matplotlib was utilized to graphically represent the optimal solution derived from the EA, providing a precise and illustrative figure of the results achieved.

This project used NumPy as the primary tool for numerical computations and array operations. This Python library, renowned for its efficiency and versatility, facilitated handling large datasets and complex mathematical functions, ensuring optimal performance throughout the project's various computational tasks.

Chapter 5

Solution Developed: OptiWood

The primary objective of OptiWood is to optimize the positioning of cuts on a wood board. The algorithm takes data about the required cuts and references a database of leftover boards, which not only facilitates the recycling of potential waste, but improves the efficiency of the manufacturing process.

This chapter provides a detailed analysis of the proposed solution to the CSP, explaining both guillotinable and non-guillotinable scenarios to ensure a comprehensive understanding of the methodologies used.

5.1 OptiWood Overview

The adopted algorithm is based on the NSGA-II but tailored with modifications that are appropriate for the specific challenges of the problem. Customized crossover and mutation operations are designed for each variable, while the selection process adheres to the standard NSGA-II methodology.

The choice of an EA to address the CSP is well-founded in the scientific community. This approach is particularly apt given the problem's NP-hard nature and its intensive computational demands. The problem's inherent characteristics further make EA one of the more efficacious solutions, as corroborated by extensive literature reviews.

5.2 Pre-Versions

Throughout the development of this project, multiple versions of OptiWood were implemented. Each version represents an evolutionary step towards a more efficient and optimized solution. Crucially, all of these versions were tested against the same dataset to ensure a consistent reference for comparison. In the following sections, we outline the key features and objectives of each version, supplemented by representative output images.

5.2.1 Version 1: Bottom-Left Fill Heuristic and Overlap Verification

Objective: To place cuts according to the Bottom-Left fill heuristic and verify overlap.

Explanation: The Bottom-Left Fill heuristic places each cut at the bottom-most, left-most position available, while introducing a check for overlap between cuts. If overlap is detected, the cut is repositioned according to predetermined rules. This version does not incorporate any optimization and strictly follows the heuristic.

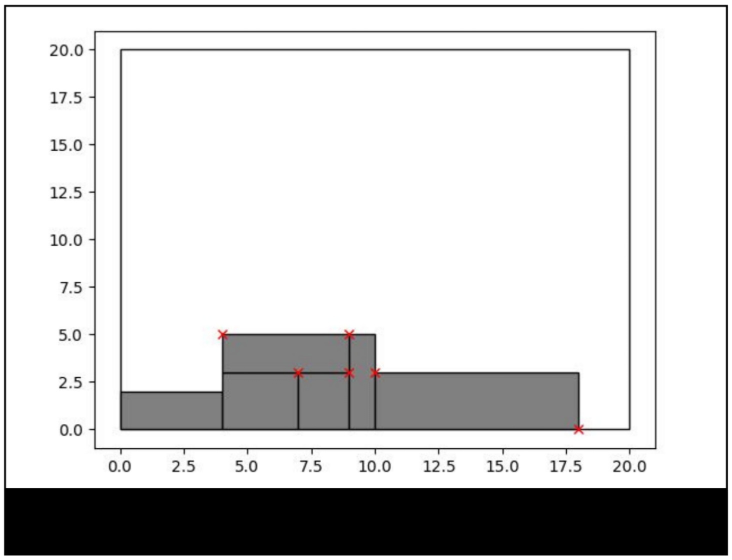


Figure 5.1: Version 1 Output

Figure 5.1 illustrates a solution that uses the bottom-left fill heuristic while checking for overlaps between pieces. In the figure, each piece is strategically placed at the lowest and most leftward available point, marked by red X's. The red X's indicate potential positions for the next piece. Once a position is utilized, it no longer appears in the figure. This method ensures that each piece is optimally placed in the bottom-left corner, thereby minimizing the height for this arrangement.

5.2.2 Version 2: Rotation

Objective: To introduce rotation of random cuts.

Explanation: By allowing rotation of random cuts, the solution space is expanded. This enhanced exploration increases the likelihood of identifying superior solutions.

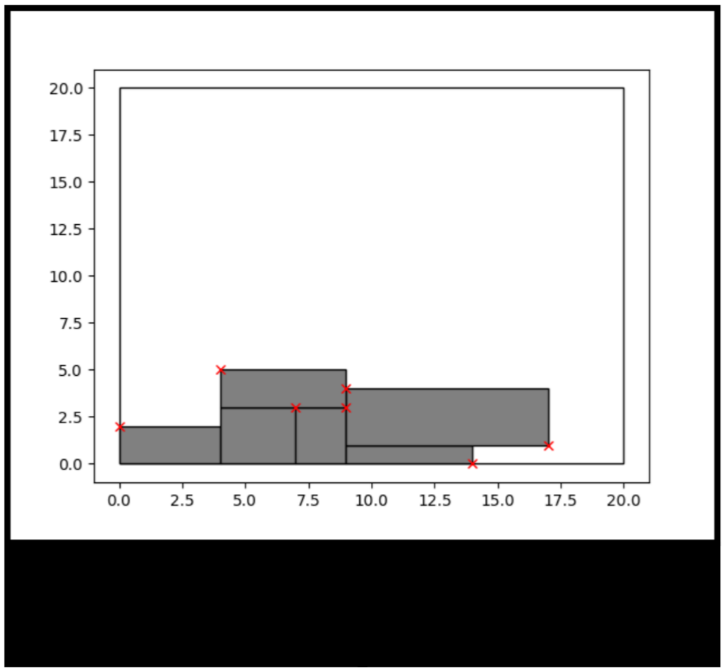


Figure 5.2: Version 2 Output

Figure 5.2 displays a solution for the same dataset presented in version 1, with the added feature of randomly rotating any cut 90º. Unlike in Figure 5.1, one of the pieces in this solution is rotated. This results in the same occupied height as before, but with a different configuration, opening up possibilities for exploring a larger solution space.

5.2.3 Version 3: Multi-Objective

Objective: To separately minimize the length and height occupied by the cuts.

Explanation: This version evolves using two objective functions to determine the best placement of the pieces; however, it does not incorporate any algorithmic optimization and strictly follows the heuristic.

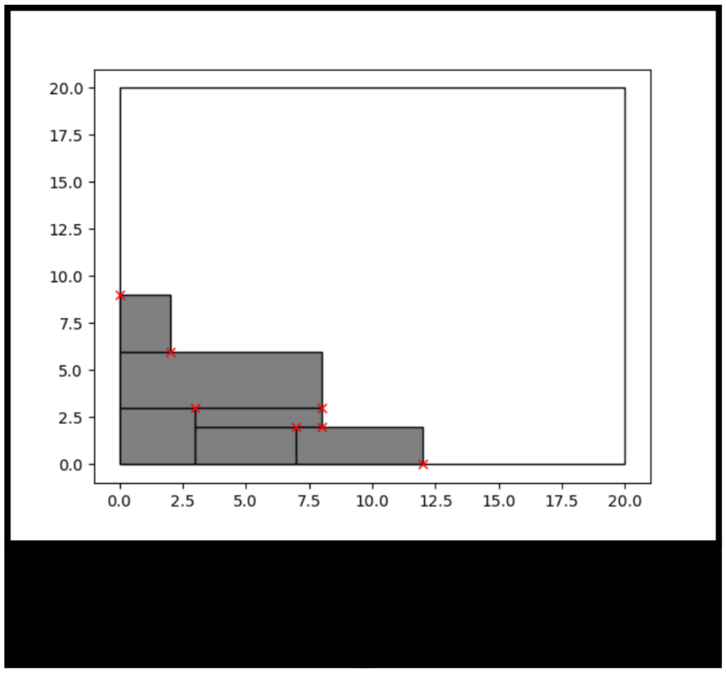


Figure 5.3: Version 3 Output

With an effective heuristic in place that includes the rotation of random pieces to broaden the solution space, this version introduces two objective functions: minimizing both the length and the height occupied. The solution in the Figure 5.3 maintains the placement of pieces in the bottom-left corner of the wood board. However, the heuristic strategy now also aims to prevent pieces from expanding too much horizontally. This approach utilizes the height of the board more efficiently, leading to a more compact solution. The updated heuristic focuses on positioning the pieces as close as possible to the bottom-left corner of the board, corresponding to the Cartesian coordinates $(0, 0)$, optimizing the solution for the problem at hand.

5.2.4 Version 4: Multi-Objective Optimization

Objective: To optimize placement using EA.

Explanation: This version introduces the optimization part of the problem, using EA, with the help of the heuristic methods from previous versions to optimize the cutting.

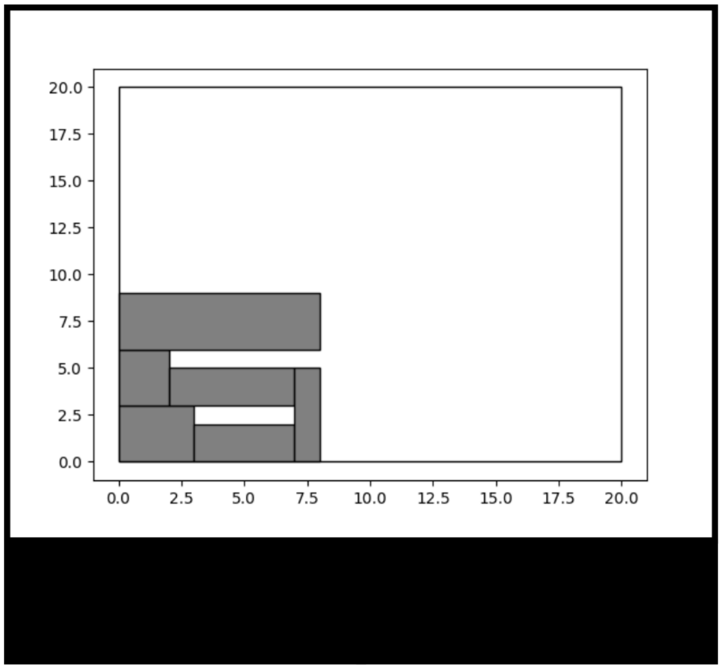


Figure 5.4: Version 4 Output

This iteration marks the initial application of the EA, specifically addressing the non-guillotinable problem. As demonstrated in Figure 5.4, the pieces are now arranged in a manner that maximizes compactness and material conservation. In this approach, only one piece is rotated randomly. Although this may not be the ideal configuration, it represents significant progress toward an optimal solution. This version continues to employ the same two objective functions as its predecessor. The evident improvement in the quality of the solution highlights the effective implementation of the EA, resulting in a considerably superior outcome compared to previous versions.

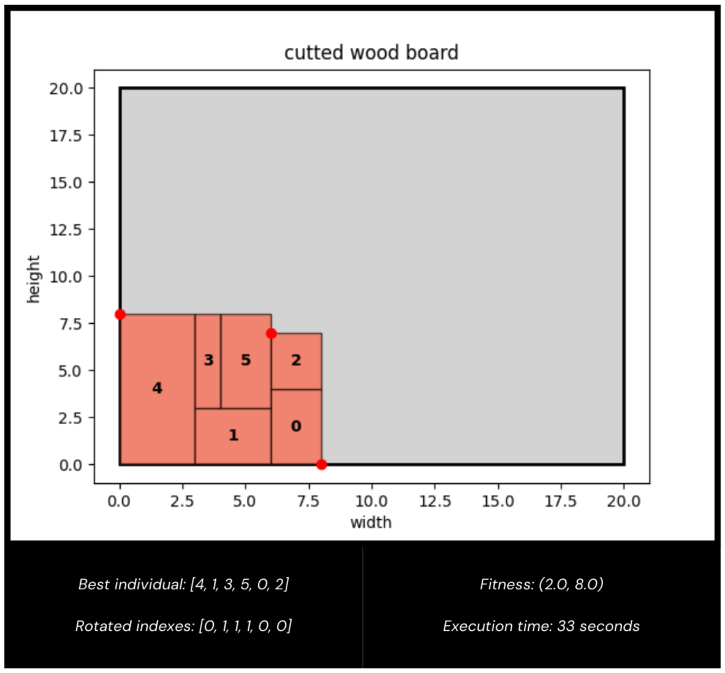
5.3 OptiWood

OptiWood represents the final version of the algorithm's development, offering two distinct algorithms tailored for guillotine and non-guillotine problems. The non-guillotinable problem algorithm evolves from previous versions, refining its objective functions and fine-tuning the parameters of the EA to achieve an optimal solution. Meanwhile, the guillotinable problem algorithm builds upon the non-guillotinable version by adding the constraint of cut lines. This addition necessitates changes in the objective functions, although it maintains the same optimized parameters as those used for the non-guillotinable problem.

5.3.1 Non-guillotine problem

In Figure 5.5, an example output is shown using the same dataset as the previous versions, but now it also displays the best individual, fitness value, and execution time. This solution is noticeably optimal, as evidenced by the absence of 'holes' in the placement sequence. Unlike earlier versions, where only one piece could rotate randomly, all pieces in this version are capable of rotation, contributing to an ideal solution. Figure 5.5 also demonstrates effective optimization of board dimensions. This is achieved through two distinct objectives: $k_1(x)$ and $k_2(x)$. Here, k_1 quantifies the difference between the envelope area and the actual area occupied by the pieces, while k_2 measures the median utilization of the board height (h) and length (l), leading to a solution that is compact and optimizes material usage.

The output of the solution includes the Best Individual, $[4, 1, 3, 5, 0, 2]$, along with the Rotated Indexes, $[0, 1, 1, 1, 0, 0]$, and the Fitness values $(2.0, 8.0)$. This configuration resulted in $k_1(x) = 2.0$ and $k_2(x) = 8.0$. In other words, the difference between the envelope area and the actual area occupied by the pieces is 2.0, and the median utilization of the board dimensions is 8.0.



Best individual: [4, 1, 3, 5, 0, 2]

Rotated indexes: [0, 1, 1, 1, 0, 0]

Fitness: (2.0, 8.0)

Execution time: 33 seconds

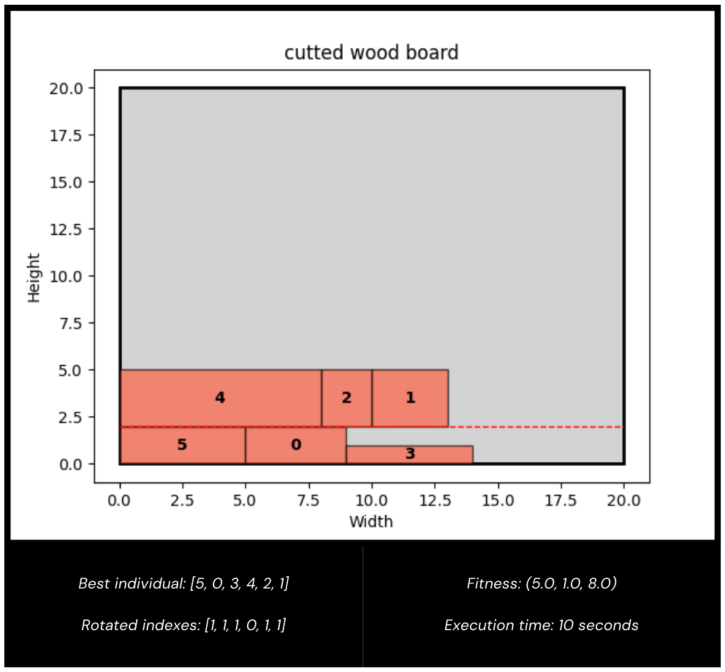
Figure 5.5: Version 5 - No-Guillotine Output

5.3.2 Guillotine problem

In Figure 5.6, the example output is displayed using the same dataset as in Figure 5.5, including the best individual, fitness value, and execution time. However, this version incorporates the constraint of cut lines characteristic of the guillotinable problem. The solution shown is markedly optimal, using only two cut lines to shape all the pieces while maximizing the height efficiency for the cuts. Figure 5.6 adeptly demonstrates the optimization of both the placement and rotation of the pieces. This efficiency is achieved through the application of three objective functions: $f_1(x)$, $f_2(x)$ and $f_3(x)$. Here, f_1 quantifies the height (h) used in the placement, f_2 evaluates the vertical misalignment of the pieces, and f_3 is analogous to k_1, representing the difference between the envelope area and the actual area occupied by the pieces.

The output of the solution includes the Best Individual, $[5, 0, 3, 4, 2, 1]$, along with the Rotated Indexes, $[1, 1, 1, 0, 1, 1]$, and the Fitness values $(5.0, 1.0, 8.0)$. This configuration results in $f_1(x) = 5.0$, indicating that the height utilized by the arrangement is 5.0,

$f_2(x) = 1.0$, representing that the vertical misalignment is 1.0, and $f_3(x) = 8.0$, measuring the difference between the envelope's area and the actual area occupied by the pieces is 8.0.



Best individual: [5, 0, 3, 4, 2, 1]

Rotated indexes: [1, 1, 1, 0, 1, 1]

Fitness: (5.0, 1.0, 8.0)

Execution time: 10 seconds

Figure 5.6: Version 5 - Guillotine Output

Chapter 6

Results

This thesis has explored the application of EA to solve the 2D CSP, focusing on both guillotine and non-guillotine cutting problems. The purpose of this chapter is to present and discuss the results obtained from the experimental phase, providing valuable insights into the effectiveness, efficiency, and applicability of the proposed EA in a real-world context.

6.1 Experimental Setup

6.1.1 Evolutionary Algorithm Parameters

During the implementation of the EA for this project, specific parameters were meticulously chosen through a series of experimental trials to ensure optimal performance and results. The parameters are as follows:

- **Population Size**: A population size of 100 was selected. This size balances computational efficiency and a diverse solution space, allowing for a comprehensive exploration of potential solutions.

- **Lambda**: A value of 200 (double the population) was chosen for lambda, determining the number of children to produce at each generation.

- **Crossover Rate**: A crossover rate of 0.1 was chosen. This relatively low rate ensures that the algorithm prioritizes exploring new regions in the solution space over exploiting existing solutions.

- **Mutation Rate**: A mutation rate of 0.85 was established. This high mutation rate emphasizes the algorithm's focus on exploring outside the current solution space, leading to the discovery of potentially superior solutions.

- **Maximum Generations**: The algorithm was set to run for a maximum of 1000 generations, providing ample opportunity for convergence toward an optimal solution.

- **Termination Criteria**: The algorithm was designed to terminate if no improvement is found to the best solution in 25 consecutive generations. This criteria ensures computational efficiency by stopping the algorithm once it is evident that further generations are unlikely to yield better solutions.

All of these parameter values were not arbitrarily chosen. However, they resulted from rigorous experimentation, ensuring that they are well suited for the problem and contribute to the algorithm's overall efficacy.

6.1.2 PC Specifications

- Operating System - Microsoft Windows 11 Home

- Processor - Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz

- Graphics Card - NVIDIA GeForce GTX 1650 (4.0 GB) / Intel(R) UHD Graphics (1.0 GB)

- Memory (RAM) - 7.8 GB Total

- Storage Drive - 475.7 GB Total

- Motherboard - CML (Stonic_CMS)

6.2 Data Presentation

To evaluate the performance of the EA implemented, three distinct datasets comprising randomly created rectangles (Tables 6.1 to 6.3) were subjected to multiple runs. The algorithm was executed 100 times on each dataset to thoroughly scrutinize its performance.

Table 6.1: Dimensions of the 6-piece dataset

	Rectangle					
Width	4	3	2	1	8	5
Height	2	3	3	5	3	2

Table 6.1 presents the dimensions of a 6-piece dataset comprising various rectangular shapes. In this dataset, the widths of the rectangles range from 1 to 8 units, while their heights vary from 2 to 5 units. The dataset features rectangles with unique dimensions, each appearing only once, thus providing a diverse range of sizes for the algorithm to process and optimize.

Table 6.2: Dimensions of the 12-piece dataset

	Rectangle											
Width	5	4	2	3	2	2	2	5	3	5	3	3
Height	5	2	4	2	5	3	4	2	5	3	2	5

Table 6.2 details a 12-piece dataset, featuring rectangles with widths ranging from 2 to 5 units and heights from 2 to 5 units. This dataset includes various sizes and shapes, such as square shapes (such as $(5, 5)$) and various rectangular forms such as $(2, 4)$ and $(4, 2)$, highlighting their distinct orientations. The repetition of certain dimensions, such as (2,5) and (5,3), emphasizes the need for multiple instances of similar-sized pieces, adding complexity to the packing optimization task.

Table 6.3: Dimensions of the 18-piece dataset

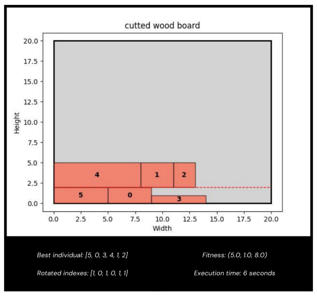
	Rectangle																	
Width	5	2	5	4	2	2	4	3	2	2	5	2	2	4	5	4	2	3
Height	3	2	5	5	2	5	5	2	2	2	5	3	2	4	4	4	2	4

Table 6.3 introduces the 18-piece dataset, the rectangle dimensions range from widths of 2 to 5 units and corresponding heights, creating a diverse selection of sizes. Notable repetitions in dimensions occur, such as the rectangle size $(5, 3)$ appearing twice and $(2, 5)$ occurring three times, among others. This repetition of certain sizes, along with the possibility of rotation, increases the complexity of the packing problem.

6.3 6-Piece Dataset Results

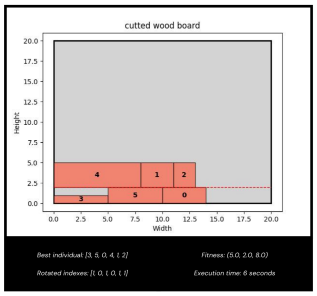
Guillotine Version

- **Overall Performance**: The algorithm achieved an optimal solution in 92% of the runs and converged to 8 unique configurations.
- **Height Optimization Focus**: The algorithm produced solutions with optimal height in 100% of the runs, resulting in 15 unique configurations.
- **Computational Time**: The average time across all runs was 8.53 seconds (standard deviation: 1.83 seconds). For optimal solutions, the average time was 8.66 seconds (standard deviation: 1.80 seconds).



Best individual: [5, 0, 3, 4, 1, 2]
Fitness: (5.0, 1.0, 8.0)
Rotated indexes: [1, 0, 1, 0, 1, 1]
Execution time: 6 seconds

Figure 6.1: 6pcs Optimal Solution

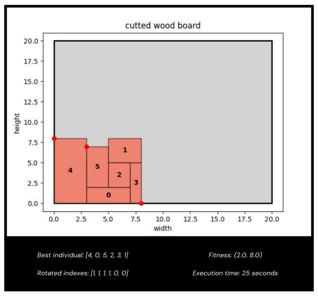


Best individual: [3, 5, 0, 4, 1, 2]
Fitness: (5.0, 2.0, 8.0)
Rotated indexes: [1, 0, 1, 0, 1, 1]
Execution time: 6 seconds

Figure 6.2: 6pcs Near-Optimal Solution

No-Guillotine Version

- **Overall Performance**: The algorithm found the optimal solution in 100% of the runs and converged to 95 unique configurations.
- **Computational Time**: The average time across all runs was 25.73 seconds (standard deviation: 0.58 seconds). Given that all runs achieve 100% optimization, this duration corresponds precisely to the average time for optimal solutions.



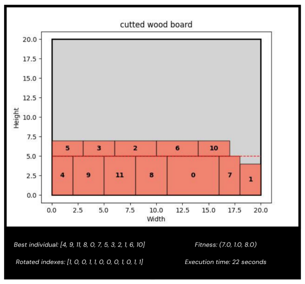
```
Best individual: [4, 0, 5, 2, 3, 1]        Fitness: (2.0, 8.0)
Rotated indexes: [1, 1, 1, 1, 0, 0]        Execution time: 25 seconds
```

Figure 6.3: 6pcs Optimal Solution

6.4 12-Piece Dataset Results

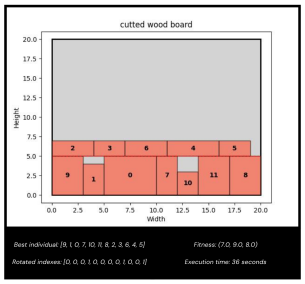
Guillotine Version

- **Overall Performance**: The algorithm achieved an optimal solution in 15% of the runs and converged to 15 unique configurations.
- **Height Optimization Focus**: Given the industrial relevance of guillotine wood cutting, it is crucial to minimize the height of the wood board. The algorithm produced solutions with optimal height in 77% of the runs, resulting in 77 unique configurations.
- **Computational Time**: The average time across all runs was 26.07 seconds (standard deviation: 9.82 seconds). For optimal solutions, the average time was 33.47 seconds (standard deviation: 14.80 seconds).



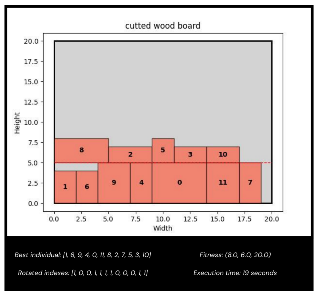
```
Best individual: [4, 9, 11, 8, 0, 7, 5, 3, 2, 1, 6, 10]    Fitness: (7.0, 1.0, 8.0)
Rotated indexes: [1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1]     Execution time: 22 seconds
```

Figure 6.4: 12pcs Optimal Solution



```
Best individual: [9, 1, 0, 7, 10, 11, 8, 2, 3, 6, 4, 5]    Fitness: (7.0, 9.0, 8.0)
Rotated indexes: [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1]     Execution time: 36 seconds
```

Figure 6.5: 12pcs Near-Optimal Solution

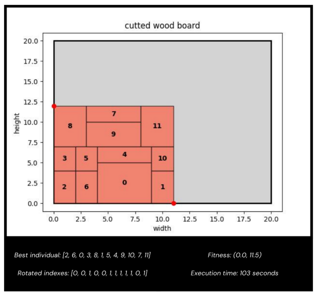


```
Best individual: [1, 6, 9, 4, 0, 11, 8, 2, 7, 5, 3, 10]    Fitness: (8.0, 6.0, 20.0)
Rotated indexes: [1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1]     Execution time: 19 seconds
```

Figure 6.6: 12pcs Non-Optimal Solution

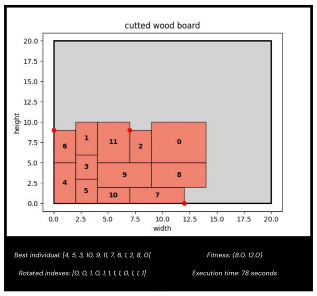
No-Guillotine Version

- **Overall Performance**: The algorithm found the optimal solution in 38% of the runs and converged to 38 unique configurations.
- **Objective Analysis**: The ideal outcome for the given dataset is a solution where the difference between the envelope area and the exact area is 0, and the average utilization of the board's height and width is 11.5. The algorithm only yielded fitness values of either (0.0, 11.5) or (8.0, 12.0).
- **Computational Time**: The average time across all runs was 99.43 seconds (standard deviation: 26.47 seconds). For optimal solutions, the average time was 114.61 seconds (standard deviation: 31.97 seconds).



Best individual: [2, 6, 0, 3, 8, 1, 5, 4, 9, 10, 7, 11]
Fitness: (0.0, 11.5)
Rotated indexes: [0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1]
Execution time: 103 seconds

Figure 6.7: 12pcs Optimal Solution



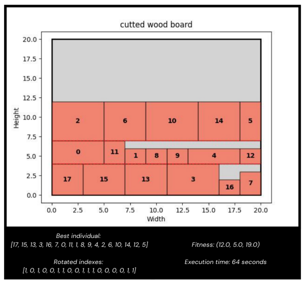
Best individual: [4, 5, 3, 10, 9, 11, 7, 6, 1, 2, 8, 0]
Fitness: (8.0, 12.0)
Rotated indexes: [0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1]
Execution time: 78 seconds

Figure 6.8: 12pcs Near-Optimal Solution

6.5 18-Piece Dataset Results

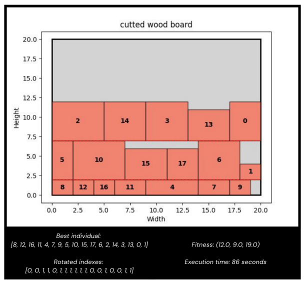
Guillotine Version

- **Overall Performance**: The algorithm achieved an optimal solution in 1% of the runs and converged to 1 unique configuration.
- **Height Optimization Focus**: The algorithm produced solutions with optimal height in 20% of the runs, resulting in 20 unique configurations.
- **Computational Time**: The average time across all runs was 42.91 seconds (standard deviation: 18.07 seconds). The time to find the only optimal solution was 64 seconds.



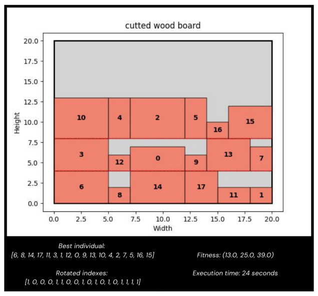
Best individual:
[17, 15, 13, 3, 16, 7, 0, 11, 1, 8, 9, 4, 2, 6, 10, 14, 12, 5]
Fitness: (12.0, 5.0, 19.0)
Rotated indexes:
[1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
Execution time: 64 seconds

Figure 6.9: 18pcs Optimal Solution



Best individual:
[8, 12, 16, 11, 4, 7, 9, 5, 10, 15, 17, 6, 2, 14, 3, 13, 0, 1]
Fitness: (12.0, 9.0, 19.0)
Rotated indexes:
[0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1]
Execution time: 86 seconds

Figure 6.10: 18pcs Near-Optimal Solution

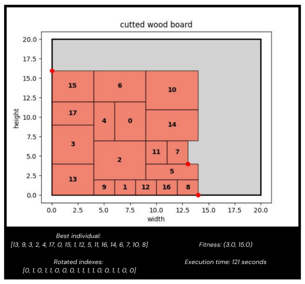


Best individual:
[6, 8, 14, 17, 11, 3, 1, 12, 0, 9, 13, 10, 4, 2, 7, 5, 16, 15]
Fitness: (13.0, 25.0, 39.0)
Rotated indexes:
[1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1]
Execution time: 24 seconds

Figure 6.11: 18pcs Non-Optimal Solution

No-Guillotine Version

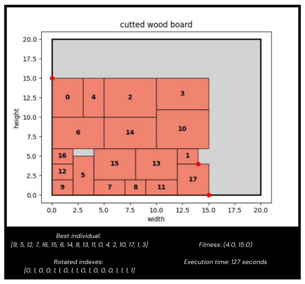
- **Overall Performance**: The algorithm found the optimal solution in 1% of the runs and converged to 1 unique configuration.
- **Objective Analysis**: The first objective measures the difference between the envelope area and the exact area, with an optimal solution being 3 and a near-optimal being 4. This represents 12% of the solutions. The second objective assesses the median utilization of the board width and height, with values ranging from an optimal 15 to a near-optimal 15.5.
- **Computational Time**: The average time across all runs was 146.20 seconds (standard deviation: 44.33 seconds). For optimal solutions, the average time was 170.33 seconds (standard deviation: 46.96 seconds).



```
Best individual:
[13, 9, 3, 2, 4, 17, 0, 15, 1, 12, 5, 11, 16, 14, 6, 7, 10, 8]    Fitness: (3.0, 15.0)

Rotated indexes:                                                   Execution time: 121 seconds
[0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0]
```

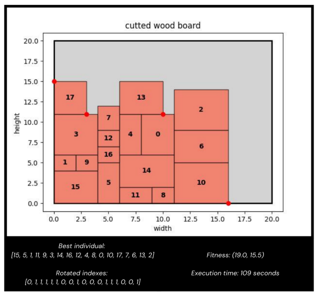
Figure 6.12: 18pcs Optimal Solution



```
Best individual:
[9, 5, 12, 7, 16, 15, 6, 14, 8, 13, 11, 0, 4, 2, 10, 17, 1, 3]    Fitness: (4.0, 15.0)

Rotated indexes:                                                   Execution time: 127 seconds
[0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1]
```

Figure 6.13: 18pcs Near-Optimal Solution



```
Best individual:
[15, 5, 1, 11, 9, 3, 14, 16, 12, 4, 8, 0, 10, 17, 7, 6, 13, 2]    Fitness: (19.0, 15.5)

Rotated indexes:                                                   Execution time: 109 seconds
[0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1]
```

Figure 6.14: 18pcs Non-Optimal Solution

6.6 Results Discussion

The performance of the EA across different datasets of rectangles offers a plethora of information on its strengths and potential areas of improvement.

In Tables 6.4 and 6.5 the following information is presented.

- **Dataset:** Refers to the size of the dataset used in the test.
- **Opt. %:** Percentage of runs in which the optimal solution was achieved.
- **Hei. %:** Percentage of runs achieving optimal height utilization.
- **N. Sol.:** Total number of unique configurations or solutions obtained in all runs.
- **Time(s):** Average computational time taken in all runs.
- **Std. Dev.(s):** Standard deviation of the computational time in runs.
- **Opt. Time(s):** Average time taken to achieve an optimal solution.

Table 6.4: Performance Metrics: Guillotine Approach

Dataset	**Opt. %**	**Hei. %**	**N. Sol.**	**Time**	**Std. Dev.**	**Opt. Time**
6-piece	92	100	8	8.53	1.83	8.66
12-piece	15	77	15	26.07	9.82	33.47
18-piece	1	20	1	42.91	18.07	64.00

Table 6.5: Performance Metrics: No-Guillotine Approach

Dataset	**Opt. %**	**N. Sol.**	**Time**	**Std. Dev.**	**Opt. Time**
6-piece	100	95	25.73	0.58	25.73
12-piece	38	38	99.43	26.47	114.61
18-piece	1	1	146.20	44.33	170.33

6.6.1 6-Piece Dataset

Guillotine Version: When faced with a smaller dataset, the algorithm showcased its efficiency and adeptness, achieving an impressive 92% success rate for optimal solutions and a perfect score for height optimization. These results attest to the algorithm's prowess with less complicated problems.

No-Guillotine Version: The algorithm's performance was stellar, converging to the optimal solution in every run. With the discovery of 95 unique optimal configurations, it is evident that the EA can navigate the solution space expertly when handling simpler datasets.

6.6.2 12-Piece Dataset

Guillotine Version: Building on the momentum of the 6-piece dataset, the algorithm still displayed commendable performance with a moderate success rate of 15% in obtaining globally optimal solutions. The diversity in the generated solutions is laudable, given the 15 unique optimal configurations discovered. The emphasis on height optimization, a critical industrial concern, saw a robust success rate of 77%, suggesting the algorithm's ability to adapt to focused optimization goals.

No-Guillotine Version: While the success rate to achieve optimal solutions was 38%, the versatility of the algorithm was evident through the 38 unique optimal configurations. Consistent fitness values across runs underline the stability and reliability of the algorithm.

6.6.3 18-Piece Dataset

As the complexity increased due to the larger number of pieces, the algorithm's challenges became more pronounced. Both the Guillotine and No-Guillotine versions achieved a low success rate of 1%, reflecting the intricacies introduced by the larger dataset. However, there is a silver lining in the board dimension utilization results. Although area optimization may not always be on point, 100% of the solutions were optimal or near-optimal with respect to board dimension utilization, indicating the algorithm's resilience.

Chapter 7

Conclusions and Future Work

The EA performance evidently scales with the complexity of the problem. It excels with smaller datasets, achieving exemplary results. However, as the size of the problem grows, there is a noticeable drop in efficiency. Despite this, its consistent performance in specific optimization objectives, such as height optimization, remains a testament to its potential. Enhancing algorithmic parameters or introducing more sophisticated genetic operators could target better performance with larger datasets.

7.1 Implications

- **Operational Efficiency in Industry**

  The superior performance of the EA, especially for smaller datasets, suggests its applicability in real-world scenarios where the number of pieces to optimize is relatively limited. This could be particularly beneficial for small to medium enterprises (SMEs) in industries such as woodworking or metal sheet cutting, where resources are limited and efficiency is crucial. The focus on height optimization resonates with the industry's concern for minimizing material wastage, implying potential cost savings.

- **Scalability Concerns**

  The decreased efficiency with larger datasets, as evidenced by the 18-piece dataset results, raises concerns about the algorithm's scalability. For larger industries or applications where numerous pieces require optimization simultaneously, the algorithm, in its current form, may not be the most effective solution. This underscores the need for further research and refinement to enhance the algorithm's scalability.

- **Diverse Solutions for Flexible Applications**

  The diversity of solutions generated by the algorithm, especially for the 6 and 12-piece datasets, offers flexibility. Different configurations can cater to varying operational needs or constraints. Multiple optimal solutions can be invaluable in real-world scenarios.

- **Potential for Hybrid Approaches**

  Given the challenges of larger datasets, there may be an opportunity to explore hybrid algorithms. Combining the EA with other optimization techniques might yield better results for more complex problems. Consistent performance in board dimension utilization, even in the 18-piece dataset, suggests that the EA has foundational strengths that can be built on.

7.2 Summary

This chapter delved into the performance evaluation of an EA designed for rectangle optimization. Key findings include:

- **High Efficiency with Smaller Datasets**: The algorithm showed optimal performance with the 6-piece dataset, achieving near-perfect results.

- **Moderate Performance for Mid-sized Datasets**: With the 12-piece dataset, while the results were admirable, there was a discernible gap between the algorithm's performance and perfect optimization.

- **Challenges with Larger Datasets**: The 18-piece dataset posed significant challenges, with the algorithm's efficiency dropping.
- **Consistency in Certain Objectives**: Despite the variation in overall performance, the algorithm consistently optimized specific objectives, such as height optimization, across all datasets.
- **Diverse Solutions**: The algorithm's ability to provide a variety of optimal configurations, especially for smaller datasets, is proof of its versatility.

7.3 Future Work

- **Enhancing Scalability**: Targeted research on improving the algorithm's performance with larger datasets can make it more universally applicable.
- **Adaptive Heuristics:** The current algorithm employs a fixed heuristic for cut placement. Adaptive heuristics could allow the system to learn from past cutting operations and fine-tune its decisions. This self-tuning feature would allow the algorithm to adapt to various types of materials and cutting scenarios, providing optimized solutions more efficiently.
- **Machine Learning Techniques:** While the genetic algorithm approach provides robust optimization, machine learning models could be incorporated to predict optimal cutting patterns based on historical and real-time data. This predictive capability could significantly reduce the computational time for optimization, making the algorithm faster and more efficient.
- **Real-Time Optimization:** The algorithm, as it stands, operates in batch mode where all data must be available upfront. Integrating real-time data could allow the algorithm to be more flexible, adapting to changes in material availability, machine conditions, or even market demand. This real-time optimization could make the algorithm invaluable for dynamic manufacturing environments.

- **Parallel Computing:** The current version of the algorithm is designed for a single-threaded operation. Using parallel computing methods would enable the algorithm to distribute its workload across multiple processors, drastically reducing computational time, and opening up possibilities for real-time or near-real-time applications, especially for large-scale industrial operations.

- **Adaptive Parameter Tuning**: Instead of static parameters, adaptive methods could be explored that dynamically change parameters based on the state of the problem.

- **Energy Efficiency:** Sustainability is an increasingly important consideration in manufacturing. Incorporating an energy efficiency objective into the algorithm could contribute to more sustainable operations by optimizing not only for material use, but also for the energy consumption of the cutting machinery.

7.4 Conclusion

This work presents a novel approach for optimizing rectangle cutting in the woodworking industry by leveraging the power of EA. The study of EA in this context has produced insightful findings that highlight both the strengths and limitations of the algorithm in various scenarios. While demonstrating notable efficiency in certain aspects of the problem, it also encounters specific challenges that require further research and refinement.

Bibliography

[1] Balcar, Š., Pilát, M., Neruda, R.: An evolutionary algorithm for 2d semi-guillotinable circular saw cutting. In: 2012 IEEE Congress on Evolutionary Computation. pp. 1–5 (2012). https://doi.org/10.1109/CEC.2012.6256455

[2] Beyaz, M., Dokeroglu, T., Cosar, A.: Robust hyper-heuristic algorithms for the offline oriented/non-oriented 2d bin packing problems. Applied Soft Computing **36**, 236–245 (2015). https://doi.org/10.1016/j.asoc.2015.06.063, `https://www.sciencedirect.com/science/article/pii/S1568494615004561`

[3] Brubach, B., Sankararaman, K.A., Srinivasan, A., Xu, P.: Algorithms to approximate column-sparse packing problems. ACM Trans. Algorithms **16**(1) (nov 2019). https://doi.org/10.1145/3355400

[4] Cintra, G., Miyazawa, F., Wakabayashi, Y., Xavier, E.: A note on the approximability of cutting stock problems. European Journal of Operational Research **183**(3), 1328–1332 (2007). https://doi.org/10.1016/j.ejor.2005.09.053, `https://www.sciencedirect.com/science/article/pii/S0377221706003079`

[5] Evtimov, G., Fidanova, S.: 2D Optimal Cutting Problem, pp. 33–39. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-65530-7_4

[6] Evtimov, G., Fidanova, S.: Heuristic algorithm for 2d cutting stock problem. In: Lirkov, I., Margenov, S. (eds.) Large-Scale Scientific Computing. pp. 350–357. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-73441-5_37

[7] Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research **13**, 2171–2175 (jul 2012)

[8] Gillies, S., van der Wel, C., van den Bossche, J., Taves, M.W., Arnott, J., Ward, B.C., et al.: Shapely. `https://github.com/shapely/shapely` (2023). https://doi.org/10.5281/zenodo.5597138, `https://pypi.org/project/Shapely`

[9] Harris, C.R., Millman, K.J., van der Walt, S.J., et al.: Array programming with NumPy. Nature **585**(7825), 357–362 (Sep 2020). https://doi.org/10.1038/s41586-020-2649-2

[10] Herring, J.R., Ed.: Opengis implementation specification for geographic information - simple feature access - part 1: Common architecture (Oct 2006)

[11] Hunter, J.D.: Matplotlib: A 2D graphics environment. Computing in Science & Engineering **9**(3), 90–95 (2007). https://doi.org/10.1109/MCSE.2007.55

[12] Katoch, S., Chauhan, S.S., Kumar, V.: A review on genetic algorithm: past, present, and future. Multimedia tools and applications **80**(5), 8091–8126 (2021). https://doi.org/10.1007/s11042-020-10139-6

[13] Van Rossum, G., Drake, F.L.: Python 3 Reference Manual. CreateSpace, Scotts Valley, CA (2009)

Appendix A

Application of 2D Packing Algorithms to the Woodwork Industry

Tiago Ribeiro[0000−0002−8939−7571], João Paulo Coelho[0000−0002−0859−8364], and Ana I. Pereira[0000−0003−3803−2043]

Research Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Bragança, 5300-253, Portugal
a40309@alunos.ipb.pt {jcoelho,apereira}@ipb.pt

Abstract. The project's objective is to solve the well-known and well-studied problem of 2D Packing applied to the Wood Work industry, in order to reduce waste through a more efficient reintegration of raw materials into the production cycle, for this multi-objective strategies, evolutionary and learning processes were explored throughout the problem-solving process.

Keywords: Optimization · Combinatorial Optimization · Packing Problems · 2D Packing.

1 Introduction

The Wood Work 4.0 (WW4.0) project aims to develop new approaches to how furniture production is carried out, mainly in Small and Medium Enterprises (SMEs). As a sector that has been modernized through the introduction of new machines and new processes, how some of the internal processes are still managed is still at a very archaic stage and impacts the overall operation of the system. Thus, the WW4.0 project aims to develop new approaches that allow the total digitization of the internal processes of the furniture production chain in such a way that they are integrated into a global approach. The 2D packing merges in the WW4.0 project with the development of optimization algorithms for the scheduling of the raw material according to the reference to be produced at a given moment, with this it is intended to reduce the waste resulting from the cutting of wood, be these regular or irregular shaped cuts. That said, the project offers numerous advantages such as the reuse of raw material, less labor effort at the factory floor level in terms of the search for raw material, such as the positioning of the cut itself, and a possible and subsequent full automation of this area of work.

This paper is divided into three sections, this first section presents the introduction, section two where the problem characterization is explored, in which the 2D packing problem will be elaborated, and the third and final section where the future work is described.

2 2D Packing Problem

The problem of 2D packing is searching for the ideal sequence of packing a set of 2D objects. This problem is considered NP-complex and has a high computational cost. As already mentioned above, the objective is to optimize the positioning and use of the sheet to be cut using multi-objective strategies, evolutionary and learning processes, this way, the search for papers relevant to the work was carried out.

Zhao et al. (Zhau et al, 2022) [3] present a learning method to solve the 2D packing problem with 2D rectangular objects. The solution is represented by the sequence of objects and the layout is built sequentially piece by piece. Centroid positioning rule techniques are explored with a lower value for the positioning of the part, then the Q-learning method is applied. Three groups of conditions are defined for the test, the computational results show that the Q-learning approach produces better compactness compared to the stochastic sequence in the layout of the parts.

Gomez and Terashima-Marin (2017) [2] propose three multi-objective evolutionary algorithms to identify sets of hyperheuristics to approximate the Pareto front. Namely Nondominated Sorting Genetic Algorithm-II, Strength Pareto Evolutionary Algorithm, and Generalized Differential Evolution Algorithm. The algorithms were extensively explored in a large set of 2D Packing problems with convex and non-convex irregularly shaped objects under different conditions and configurations. The study presents an analysis where the robustness and flexibility of the strategies outlined are evaluated, obtaining encouraging results compared with a set of simple heuristics usually used in this type of problem.

Fang et al (2021) [1] propose a strategy based on the Particle Swarm Optimization algorithm where he obtained efficient results with a reduced execution time. Hybrid strategies have been explored with promising results.

3 Future Work

To complete the paper, the main strategy will be the analysis and resolution of each of the topics:

1. Create a matrix representation of the wooden board to be cut.
2. Identify the objects to be cut in that matrix and optimize their positioning using optimization tools such as genetic algorithm and particle swarm optimization, among others.
3. Create a database to store all information regarding the leftovers (waste storage represents leftover boards that have already been cut, but can be used for some other cuts, reducing waste).
4. Explore multi-objective strategies and measures to assess the robustness and efficiencies of the strategies developed.
5. Optimize the procedure for identifying boards located in the leftovers regarding information like area, dimensions, thickness, material, and direction of the wood veins.

6. Explore the behavior of optimization algorithms in solving 2D packaging problems and incorporate learning strategies to predict parts destined for permanent waste.

Future work includes the performance evaluation of the developed algorithms using real data.

Acknowledgement

This work was supported by Norte 2020 Portuguese Program under the grant Norte-01-0247-FEDER-072593 and FCT — Fundação para a Ciência e Tecnologia within the Project Scope UIDB/05757/2020.

References

1. Fang, J., Rao, Y., Liu, P., Zhao, X.: Sequence transfer-based particle swarm optimization algorithm for irregular packing problems. IEEE Access **9**, 131223–131235 (2021)
2. Gomez, J.C., Terashima-Marín, H.: Evolutionary hyper-heuristics for tackling biobjective 2d bin packing problems. Genet Program Evolvable Mach **19**, 151–181 (2018). https://doi.org/10.1007/s10710-017-9301-4
3. Zhao, X., Rao, Y., Fang, J.: A reinforcement learning algorithm for the 2d rectangular strip packing problem. J. Phys.: Conf. Ser. 2181 012002 (2022)

Appendix B

Aplicação de Algoritmos de Empacotamento 2D na indústria da madeira

Tiago Ribeiro[1]; Ana I. Pereira[2]
[1]Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Portugal

tiagobribeiro@ipb.pt

Resumo

O problema do empacotamento 2D é aplicado em diversas áreas, nomeadamente no corte industrial (madeira e vidro) e embalagem (transporte e armazenagem).O objetivo deste trabalho é aplicar os conceitos do empacotamento 2D à indústria da Madeira, de forma a reduzir o desperdício através de uma reintegração mais eficiente das matérias-primas no ciclo produtivo, assim estabelece-se a criação de um algoritmo capaz de resolver um problema de empacotamento 2D com formas regulares, adaptando-o a todas as formas utilizando estratégias multi-objetivas combinadas com processos evolutivos como *particle swarm optimization* ou algoritmos genéticos explorados no contexto do problema em estudo. Os resultados são razoáveis em comparação com os resultados obtidos por M. Chen et al. no artigo "*A two-level search algorithm for 2D rectangular packing problem*", cujos resultados são obtidos através de um *greedy algorithm*.

Palavras-chave: otimização; otimização combinatória; problemas de empacotamento; empacotamento 2D

Financiamento: Este trabalho foi suportado pelo Programa Português Norte 2020 no âmbito do financiamento estratégico Norte-01-0247-FEDER-072593 e FCT — Fundação para a Ciência e Tecnologia para a finalidade do projeto UIDB/05757/2020.

Application of 2D Packing Algorithms to the woodwork industry

Tiago Ribeiro[1]; Ana I. Pereira[2]
[1]Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Portugal

tiagobribeiro@ipb.pt

Abstract

The 2D packing problem is applied in several areas, namely in industrial cutting (wood and glass) and packaging (transport and storage). The work's objective is to apply the concepts of 2D packing to the wood industry to reduce waste through a more efficient reintegration of raw materials into the production cycle. Thus, settling down an algorithm capable of solving a 2D packing problem with regular shapes and adapting it to all shapes using multi-objective strategies combined with evolutionary processes such as particle swarm optimization or genetic algorithms explored in the context of the problem under study. The results are reasonable compared to the results obtained by M. Chen et al. in the article "A two-level search algorithm for 2D rectangular packing problem," which results are obtained through a greedy algorithm.

Keywords: optimization; combinatorial optimization; packing problems; 2D packing

Funding: This work was supported by Norte 2020 Portuguese Program under the grant Norte-01-0247-FEDER-072593 and FCT — Fundação para a Ciência e Tecnologia within the Project Scope UIDB/05757/2020.

Appendix C

2D Packing in Woodwork Industry

Tiago B. Ribeiro and Ana I. Pereira

Research Centre in Digitalization and Intelligent Robotics (CeDRI), Instituto Politécnico de Bragança, Bragança, Portugal

Abstract. Adopting computational technologies can revolutionize traditional sectors through enhanced efficiency and sustainability. This study explores the use of 2D packaging algorithms in woodworking industries, which often leads to material waste, and reduces dependence on manual labor. Implementing these algorithms can optimize the use of materials, labor, and promote scalable production. This study evaluated the effectiveness of the 2D cutting algorithm in standard woodworking, highlighting its potential to reduce waste and its industrial impact, emphasizing the importance of merging technology with traditional industries and advocating a future that harmonizes economic development with sustainable resource use.

Keywords: Optimization, Multi-objective optimization, 2D Packing problems, 2D Cutting-Stock Problems.

1 Introduction

The woodworking industry, deeply entrenched in traditional practices, is on the brink of a transformative era. Historically anchored in manual labor and expertise, there is a burgeoning realization of the untapped potential of digital optimization within this sector. This contribution delves into applying two-dimensional (2D) packing algorithms in woodworking to uncover opportunities for enhanced efficiency, reduced waste, and heightened productivity.

Packing and cutting stock problems are based in computational geometry and operations research, optimization challenges centered on minimizing space utilization when fitting objects into containers. The woodworking environment, where every fragment of raw material has economic and environmental ramifications, amplifies the importance of these challenges. With their reliance on skilled labor for tasks like estimation and cutting, conventional woodworking techniques frequently encounter issues of material waste and scalability. By weaving in algorithmic strategies, there is potential to optimize material and labor resources and reimagine the boundaries of what is feasible at an industrial level.

This contribution highlights the effectiveness of 2D packing algorithms in standard woodworking tasks. The intent is to discern the potential of these algorithms to refine existing practices, especially in areas like waste mitigation and labor efficiency. Beyond the immediate operational benefits, the discussion extends to the more significant ramifications of such technological integrations, particularly emphasizing the environmental benefits that result from thoughtful use of timber.

This paper is organized as follows. Section 1 serves as the introduction, establishing the context and objectives of the research; Section 2 provides a comprehensive literature review, highlighting relevant studies; Section 3 describes the research methodology, detailing the methods and procedures employed; Section 4 presents the empirical results and their analysis; and Section 5 concludes the paper, encapsulating the primary findings and suggesting potential areas for future research.

2 Literature Review

The application of computer science to traditional industries presents exciting opportunities to improve efficiency and sustainability. One sector where this potential remains to be explored is the woodwork industry, which traditionally relies on manual labor and craftsmanship. Although these elements are crucial in the production process, they are often associated with high levels of waste and limitations in scalability. This context makes the woodworking industry ripe for applying computational techniques, such as two-dimensional (2D) packing algorithms, to improve material usage and productivity.

Beyaz et al. [2] delves into the intricacies of the offline 2D bin-packing problem (2DBPP), a well-recognized NP-hard combinatorial optimization challenge. Although several heuristic methods have been proposed in the past, exact approaches for larger problem instances have not yet been discovered. Traditional methods were successfully applied, such as next-fit, first-fit, best-fit, unified tabu search, genetic, and memetic algorithms. This study presents a novel set of meta-heuristic algorithms that ingeniously select and combine the best features of state-of-the-art heuristics and local search techniques. A standout feature of these proposed algorithms is the introduction of new crossover and mutation operators explicitly designed for heuristic selection. The robustness and efficiency of the algorithms are demonstrated through extensive experiments on benchmark problem instances for offline 2DBPP. The results are promising and showcase the algorithm's ability to consistently achieve solutions close to optimal. Brubach et al. [3] focus on the exploration of column-sparse packing problems, a subset of combinatorial optimization problems that have garnered significant attention due to their intricate nature and wide-ranging applications. These problems involve the challenge of efficiently packing items into containers, considering the sparsity of the packing matrix columns. The primary objective is to achieve an optimal or near-optimal packing solution while adhering to the constraints imposed by the sparsity. The work introduces two groundbreaking ideas in this domain: attenuation and multiple-chance algorithms. These concepts are innovative strategies to derive improved approximation algorithms for column-sparse packing problems. The attenuation approach involves adjusting the packing constraints to achieve a more balanced and efficient solution. On the other hand, multiple-chance algorithms provide a probabilistic framework, offering multiple opportunities to achieve an optimal packing configuration. Through rigorous experimentation and analysis, the authors demonstrate the efficacy of these methods in obtaining solutions that closely approximate the optimal. Balcar et al. [1] delve into a unique class of 2D stock cutting problems, precisely the semi-guillotinable problems. This classification is paramount when devising optimal cutting plans for circular saws, a standard tool in various manufacturing processes. The authors introduce a new algorithm specifically tailored for both guillotinable and non-guillotinable 2D cutting stock problems. Their research aims to bridge the gap between traditional cutting methods and the evolving demands of modern manufacturing. Their approach offers a fresh perspective on optimizing material usage, ensuring minimal waste, and maximizing efficiency in the cutting process. Evtimov and Fidanova [6, 7], address the challenges posed by the 2D cutting stock problem, especially when the items are irregular polygons. Recognizing the inherent complexity and computational challenges of the problem, the authors propose a novel stochastic algorithm as a solution. This algorithm handles the intricacies of cutting irregularly shaped items from larger stock material, ensuring optimal utilization and minimal wastage. The paper underscores the importance of developing advanced heuristic manufacturing and material optimization methods. This approach stands out for its ability to provide efficient solutions to real-world cutting problems, emphasizing the potential of heuristic algorithms to address complex industrial challenges. Cintra et al. [4] delves into the intricacies of cutting stock problems and their approximability. The authors shed light on the similarities and differences between cutting stock problems and bin-packing problems, emphasizing the variability of input items as a distinguishing factor. Furthermore, they established that the two-dimensional cutting stock problem shares the same level of approximability difficulty as its two-dimensional bin packing counterpart.

3 Methodology

This section presents the analytical techniques applied to the 2D cutting stock problem, analyzing both the guillotinable and non-guillotinable cutting methods, focusing on their respective characteristics and challenges. Furthermore, it introduces the problem's definition, followed by a comprehensive formulation that serves as the foundation for the proposed solutions.

3.1 Problem Definition

This project addresses the 2D cutting stock problem, also known as the 2D packing problem. The primary objective is to optimize the utilization of previously used stock material sourced from a database. By doing so, this study aims to reuse leftovers, thus recycling materials that might otherwise be discarded, promoting sustainable practices, and maximizing the value derived from each piece of material.

In 2D cutting stock problems, it is possible to identify two types of problem definitions: guillotine and non-guillotine.

Guillotinable Cutting Stock Problem: In the guillotinable cutting stock problem, the cuts are made sequentially, where each cut extends from one edge of the material to the opposite edge without interruptions, as shown in Figure 1. This problem ensures that the material is divided into distinct rectangular pieces needed for some of the machinery used in woodworking. The advantage of guillotinable cuts is their simplicity and efficiency; however, they may only sometimes yield the most optimal use of material, especially when the desired pieces have irregular shapes.

Non-Guillotinable Cutting Stock Problem: The non-guillotinable cutting stock problem allows for cuts that do not necessarily extend from one edge to another, providing greater flexibility in cutting patterns and accommodating more complex or irregular shapes, as shown in Figure 2. Although this problem can potentially result in a more optimized use of material, it might be more challenging to implement, especially when using automated machinery.

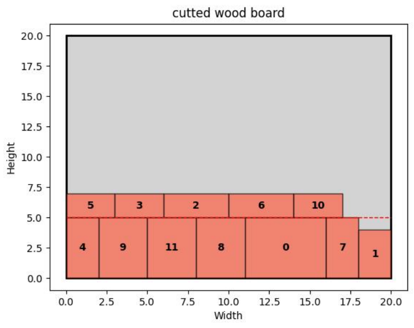


Fig. 1. Guillotine solution example

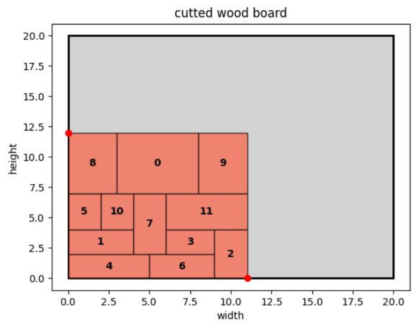


Fig. 2. Non-guillotine solution example

3.2 Problem Formulation

The problem formulation segment delves into the quantitative representation of the 2D Cutting Stock Problem. Through mathematical models and equations, it is possible to define the objective function to optimize the used material.

Mathematical formulation: Considering the problem of packing $P_1, P_2, ..., P_n$ rectangular pieces, where P_i is defined by its vertices $(v_1^i, v_2^i, v_3^i, v_4^i)$, as represented in Figure 3:

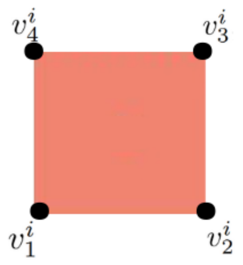


Fig. 3. Vertices Representation

Let $x = [[y_1, y_2, ..., y_n], [Y_1, Y_2, ..., Y_n]]$, where $y_i \in \{P_1, P_2, ..., P_n\}$ and $Y_i \in \{0, 1\}$ represent a possible solution to the proposed problem. For each element x, it is necessary to calculate the length used in the cut, l, the height used in the cut, h, and the vertical misalignment of all pieces, s.

Guillotine version: The objective functions are as follows:

- $f_1(x) = h$, representing the total height used in the solution.
- $f_2(x) = s$, where $s = \sum_{i=1}^{n} g(x_i)$, where $g(x_i) = \begin{cases} h_{i-1} - h_i & \text{if } h_i \leq h_{i-1} \\ 2(h_i - h_{i-1}) & \text{if } h_i > h_{i-1} \end{cases}$, this represents the sum of the vertical misalignment of piece i in relation to the previous piece.
- $f_3(x) = A_{env} - \sum_{i=1}^{n} A_i$, where A_{env} represents the envelope area, which is the length multiplied by the height occupied by the pieces $(l \times h)$ and A_i represents the area of the piece x_i.

Non-guillotine version: The objective functions are as follows:

- $k_1(x) = A_{env} - \sum_{i=1}^{n} A_i$, where A_{env} represents the envelope area and A_i represents the area of the piece x_i.
- $k_2(x) = \frac{l+h}{2}$, indicative of the average utilization of the board height and length by the pieces.

3.3 Evolutionary Algorithm

Evolutionary algorithms (EAs) are metaheuristic algorithms inspired by biological evolution and the Darwinian principle of survival of the fittest [8]. It uses chromosome representation, fitness selection, and biologically inspired operators. Chromosomes, typically in binary string format, represent solutions. The fitness function assesses each chromosome's suitability, and through operators such as selection, mutation, and crossover, the algorithm iterates on potential solutions. Specifically, crossover alters subsequences between chromosomes, while mutation may flip bits according to probability.

The NSGA-II algorithm is an adaption for multi-objective problems, where the selection operator handles the selection of solutions based on their dominance relationship in the objective space, maintaining diversity while ensuring that non-dominated solutions are prioritized [5].

4 Results

This study developed an algorithm that takes data about the required cuts and references a database of leftover boards, which not only facilitates the recycling of potential waste, but also improves the efficiency of the manufacturing process.

This section provides a detailed analysis of the proposed solution to the 2D cutting stock problem, explaining both guillotinable and non-guillotinable scenarios to ensure a comprehensive understanding of the methodologies used.

To evaluate the performance of the evolutionary algorithm implemented, three distinct datasets comprising randomly created rectangles were subjected to multiple runs. The algorithm was executed 100 times on each dataset to thoroughly scrutinize its performance.

Guillotine problem: Table 1 provides metrics on the performance of NSGA-II when using the guillotine cutting approach. The following table presents information about the results obtained:

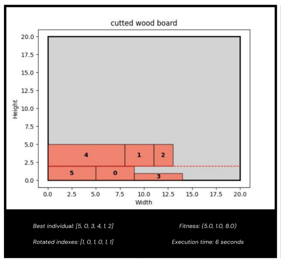
– **Dataset:** Refers to the size of the dataset used in the test.
– **Opt. %:** Percentage of runs in which the optimal solution was achieved.
– **Hei. %:** Percentage of runs achieving optimal height utilization.
– **N. Sol.:** Total number of unique configurations or solutions obtained in all runs.
– **Time(s):** Average computational time taken in all runs.
– **Std. Dev.(s):** Standard deviation of the computational time in runs.
– **Opt. Time(s):** Average time taken to achieve an optimal solution.

Table 1. NSGA-II Performance Metrics: Guillotine Approach

Dataset	Opt. %	Hei. %	N. Sol.	Time	Std. Dev.	Opt. Time
6-piece	92	100	8	8.53	1.83	8.66
12-piece	15	77	15	26.07	9.82	33.47
18-piece	1	20	1	42.91	18.07	64.00

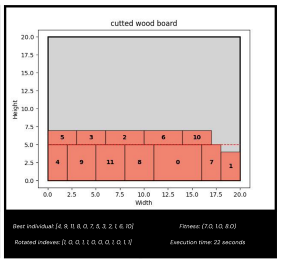
For the 6-piece dataset, the algorithm shines with a 92% optimal solution rate and 100% height optimization. It identifies eight unique configurations and averages a computational time of 8.53 seconds. The 12-piece dataset shows a reduced optimal solution rate of 15% but a strong 77% in optimal height optimization. The algorithm converges to 15 configurations, with a computational average of 26.07 seconds. Despite the complexity of the 18-piece data set, the algorithm still achieves a 1% optimal rate and a 20% optimal height optimization, with an average time of 42.91 seconds.

Some solutions obtained can be visualized in Figures 4 to 6.



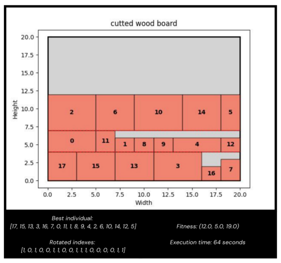
Best individual: [5, 0, 3, 4, 1, 2]
Fitness: (5.0, 1.0, 8.0)
Rotated indexes: [1, 0, 1, 0, 1, 1]
Execution time: 6 seconds

Fig. 4. 6-piece



Best individual: [4, 9, 11, 8, 0, 7, 5, 3, 2, 1, 6, 10]
Fitness: (7.0, 1.0, 8.0)
Rotated indexes: [1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1]
Execution time: 22 seconds

Fig. 5. 12-piece



Best individual:
[17, 15, 13, 3, 16, 7, 0, 11, 1, 8, 9, 4, 2, 6, 10, 14, 12, 5]
Fitness: (12.0, 5.0, 19.0)
Rotated indexes:
[1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
Execution time: 64 seconds

Fig. 6. 18-piece

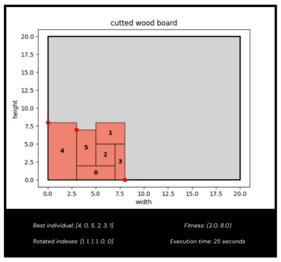
No-guillotine problem: Table 2 provides metrics for the performance of NSGA-II when using the no-guillotine cutting approach.

Table 2. NSGA-II Performance Metrics: No-Guillotine Approach

Dataset	Opt. %	N. Sol.	Time	Std. Dev.	Opt. Time
6-piece	100	95	25.73	0.58	-
12-piece	38	38	99.43	26.47	114.61
18-piece	1	1	146.20	44.33	170.33

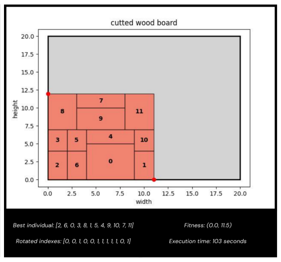
For the 6-piece dataset, the algorithm demonstrates maximum efficiency with an optimal solution rate of 100%. It pinpoints 95 unique configurations with an average computational duration of 25.73 seconds. In the 12-piece dataset, the algorithm achieves an optimal solution rate of 38% and identifies 38 unique configurations, taking an average of 99.43 seconds for computation. Examining the 18-piece data set, given its intricacies, the algorithm manages an optimal rate of 1% and discovers a single unique configuration, requiring an average computational time of 146.20 seconds.

Some solutions obtained can be visualized in Figures 7 to 9.



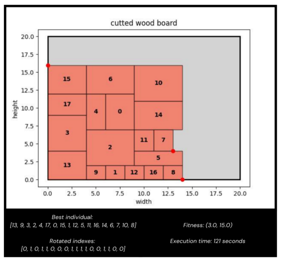
Best individual: [4, 0, 5, 2, 3, 1]
Fitness: (2.0, 8.0)
Rotated indexes: [1, 1, 1, 1, 0, 0]
Execution time: 25 seconds

Fig. 7. 6-piece



Best individual: [2, 6, 0, 3, 8, 1, 5, 4, 9, 10, 7, 11]
Fitness: (0.0, 11.5)
Rotated indexes: [0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1]
Execution time: 103 seconds

Fig. 8. 12-piece



Best individual:
[13, 9, 3, 2, 4, 17, 0, 15, 1, 12, 5, 11, 16, 14, 6, 7, 10, 8]
Fitness: (3.0, 15.0)
Rotated indexes:
[0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0]
Execution time: 121 seconds

Fig. 9. 18-piece

In conclusion, it is possible to verify the following situations.

6-Piece Dataset: The 6-piece dataset was an initial benchmark to measure NSGA-II performance. The guillotine approach was particularly notable, achieving a remarkable 92% success rate for optimal solutions. The non-guillotine approach showcased equal efficiency, converging to the optimal solution in every run. Such results underscore the algorithm's ability to deal with less complex scenarios.

12-Piece Dataset: The intricacies became more apparent as we delved into the 12-piece dataset. While experiencing a slight drop in performance, the guillotine approach still managed a 77% success rate in height optimization, emphasizing its utility in specific industrial contexts. On the other hand, the no-guillotine approach revealed its adaptability, discovering multiple unique optimal configurations despite the increased challenge.

18-Piece Dataset: The 18-piece dataset proved to be the most challenging, size and complexity led to a decline in the optimal solution success rate. However, the resilience of NSGA-II shone through, as it consistently generated solutions with optimal or near-optimal board dimension utilization. This persistence highlights the algorithm's drive for efficiency, even when faced with more considerable challenges.

5 Conclusions and Future Work

This research embarked on the journey to explore the capabilities of evolutionary algorithms in solving the 2D cutting stock problem, adopting both guillotine and no-guillotine cutting strategies. The algorithm's performance was meticulously analyzed across three distinct datasets, each reflecting different problem complexities.

In sum, while the algorithm exhibits exceptional capabilities with simpler problems, its efficiency diminishes as the complexity of the problem grows. However, its unwavering performance in targeted optimization objectives, such as height optimization, sets it apart.

Future endeavors could be **enhancing scalability** to ensure the algorithm's performance is sustained or even improved with larger datasets, enhancing its universal applicability. Another promising strategy is the exploration of **hybrid optimization approaches**; Integrating NSGA-II with other optimization techniques might produce a more robust and efficient solution, particularly for complex challenges. Further value can come from **real-world testing**. By deploying the algorithm in tangible industrial contexts, we can derive practical insights and opportunities for refinement, factoring in real operational constraints. Delving deeper into the genetic framework, the introduction of **advanced genetic operations** can elevate the algorithm's diversity and efficiency in solution generation. Lastly, shifting from static to **adaptive parameter tuning** offers a dynamic approach, with parameters evolving in response to the problem's state, which could significantly enhance the optimization process.

Acknowledgments

This work was supported by Norte 2020 Portuguese Program under the grant Norte-01-0247-FEDER-072593 and FCT — Fundação para a Ciência e Tecnologia within the Project Scope UIDB/05757/2020 and UIDP/05757/2020.

References

1. Balcar, Š., Pilát, M., Neruda, R.: An evolutionary algorithm for 2d semi-guillotinable circular saw cutting. In: 2012 IEEE Congress on Evolutionary Computation. pp. 1–5 (2012)
2. Beyaz, M., Dokeroglu, T., Cosar, A.: Robust hyper-heuristic algorithms for the offline oriented/non-oriented 2d bin packing problems. Applied Soft Computing 36, 236–245 (2015)
3. Brubach, B., Sankararaman, K.A., Srinivasan, A., Xu, P.: Algorithms to approximate column-sparse packing problems. ACM Trans. Algorithms 16(1) (2019)
4. Cintra, G., Miyazawa, F., Wakabayashi, Y., Xavier, E.: A note on the approximability of cutting stock problems. European Journal of Operational Research 183(3), 1328–1332 (2007)
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
6. Evtimov, G., Fidanova, S.: 2D Optimal Cutting Problem, pp. 33–39. Springer International Publishing (2018)
7. Evtimov, G., Fidanova, S.: Heuristic algorithm for 2d cutting stock problem. In: Lirkov, I., Margenov, S. (eds.) Large-Scale Scientific Computing. pp. 350–357. Springer International Publishing (2018)
8. Koziel, S., Michalewicz, Z.: Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. Evolutionary computation 7(1), 19–44 (1999)