OPTIMIZING EFFICIENCY IN A 1D CSP: A COMPUTATIONAL INTELLIGENCE STUDY

Innovative Ant Colony Optimization Approaches for the Cutting Stock Problem: An Experimental Comparison

Student:
ID: 190079330

Date: April 2024
Location: Birmingham – United Kingdom

Table of Contents

1 Problem Overview:

This report examines the Cutting Stock Problem (CSP) in manufacturing industries like metal fabrication and woodworking, where the aim is to optimize the cutting of stock materials into specified lengths, efficiently minimizing waste and cost while ensuring timely order fulfillment[1]. The focus is on operational strategies to enhance material utilization and meet stringent delivery schedules.

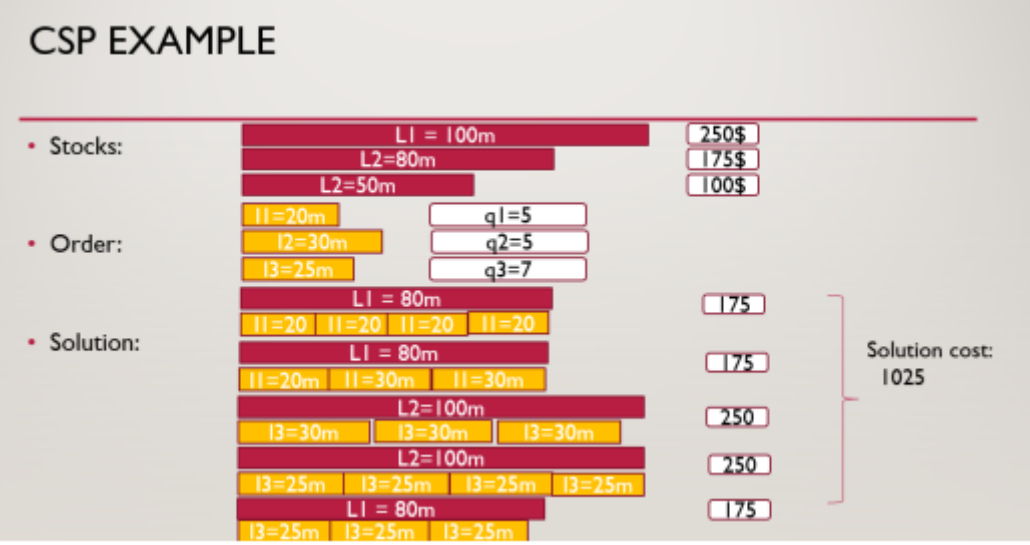


Figure 1 A Graphical CSP Example

2 Methodology

2.1 Overview of Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is a probabilistic technique derived from computational intelligence, designed to address combinatorial optimization problems. The algorithm, inspired by the foraging behaviors of ants, utilizes pheromone trails for indirect communication, directing ants towards optimal solutions (Figure 2). ACO's effectiveness is highlighted in applications such as the Traveling Salesman Problem (TSP) and the Cutting Stock Problem (CSP), where the objective is to minimize waste while optimizing the use of stock materials in manufacturing and construction. This approach has been validated by Poldi and Arenales [1], who noted its efficacy in generating near-optimal solutions, particularly in low-demand scenarios.

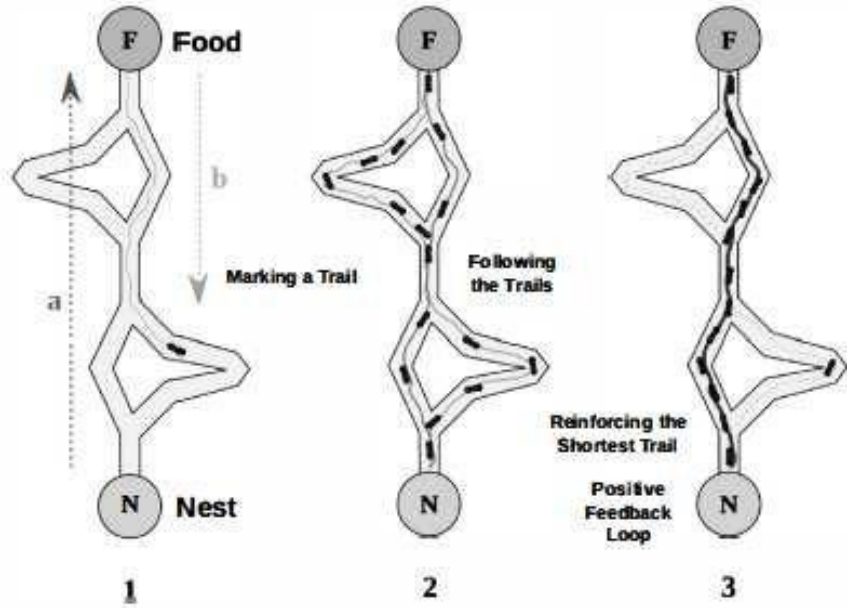


Figure 2 Functionality of ACO algorithm [7]

2.2 Mathematical Representation and Optimization of CSP with ACO

CSP involves selecting stock lengths to cut specified pieces at a minimal cost. Mathematically, it is framed as:

- Let m be the number of different stock lengths available
- Let n represent the types of pieces needed.
- $L = [l_1, l_2, \dots, l_m]$ denotes available stock lengths.
- $C = [c_1, c_2, \dots, c_m]$ denotes costs associated with each stock length.
- $P = [p_1, p_2, \dots\ p_n]$ the lengths of pieces required.
- $Q = [q_1, q_2, \dots\ q_n]$ represents the quantities of each type of piece required.

2.2.1 Solution Representation (Tours):

Each solution by an ant is a sequence of decisions detailing from which stock length each piece type should be cut. Formally, a solution S for an ant can be represented as a matrix A of dimension $m \times n$, where a_{ij}in the matrix A denotes the number of pieces of type j cut from stock type i.

Fitness Evaluation: The fitness of a Solution S is calculated as:

$$Fitness\ (S) = \sum_{i=1}^{m} x_i \times\ c_i$$

[3]

Where x_i is derived from the matrix A indicating if stock i is used in the solution.

2.3 Role of Ants and Pheromone Updates.

2.3.1 Ant Behavior:

Each ant constructs a solution by choosing stock lengths to cut pieces based on a probability influenced by pheromone concentration and heuristic information (e.g., piece fit or cost efficiency). The probability p_{ij} that an ant uses stock i to cut piece j is modeled as:

$$p_{ij} = \frac{\tau_{ij}^{a}\ \cdot\ \eta_{ij}^{\beta}}{\sum_{k=1}^{m} \tau_{kj}^{a}\ \cdot\ \eta_{kj}^{\beta}}$$

[3]

where:

- $\tau_{i\,j}$ is the pheromone concentration on using stock i for piece j,
- η_{ij} is a heuristic value (e.g., $\frac{p_i}{l_i}$ indicating how well piece j fits into stock i),
- α and β control the influence of the pheromone and the heuristic value, respectively.

2.3.2 Pheromone Update:

After all ants have constructed their solutions, pheromones are Updated to reflect the learned quality of decisions. The update rule typically follows:

$$\tau_{i\,j} \leftarrow (1-\rho)\ \cdot\ \tau_{i\,j} +\ \Delta\,\tau_{i\,j}$$

[3]

Where ρ is the pheromone decay rate, and $\Delta\,\tau_{i\,j}$ represents the amount of pheromone deposited, which could be dependent on the reduction in cost or improvement solution quality when stock i is used for piece j .

2.4 Integration in CSP

The integration of Ant Colony Optimization (ACO) for the Cutting Stock Problem (CSP) involves formulating heuristic values and a pheromone update strategy that balances solution exploration and exploitation, with dynamic adjustments to decay and mutation rates enhancing adaptability and the potential for finding globally optimal solutions [2] [3].

2.5 Solution Design

2.5.1 ACO with Dynamic Mutation

The initial solution employs an Ant Colony Optimization (ACO) algorithm augmented with a dynamic mutation mechanism to enhance solution space exploration and prevent convergence on suboptimal solutions [2]. The mutation rate is adaptively adjusted, decreasing following cost reductions to exploit promising areas, and increasing to encourage exploration otherwise. This adaptive strategy effectively balances exploration and exploitation, crucial for optimal solutions in complex optimization scenarios [2].

2.5.2 Adapted Hybric ACO with Local Search

The second approach enhances the foundational ACO model by integrating a local search mechanism and advanced pheromone update strategies [3]. Initial solutions constructed through standard ACO methods are refined by a local search algorithm, which iteratively improves solutions by exploring their immediate neighborhood for more optimal configurations [3]. Additionally, the pheromone update rule is designed to reinforce beneficial paths and is fine-tuned to decay at a rate that preserves potentially viable paths.

2.6 Experimental Setup

To evaluate the efficacy and performance of the proposed Ant Colony Optimization (ACO) solutions, computational experiments were conducted in Google Colab, a scalable environment for Python-based algorithms. Preliminary testing determined key parameters like the number of ants and pheromone decay rate, while data collection through a local spreadsheet enabled detailed analysis and comparison of metrics such as waste reduction and stock usage efficiency.

3 Implementation

3.1 ACO with Dynamic Mutation

The Ant Colony Optimization (ACO) algorithm, adapted for the Cutting Stock Problem (CSP) with multiple stock lengths, leverages dynamic mutation to minimize

material waste and costs in industrial settings, building upon the research by Lu, Wang, and Chen (2008). This implementation introduces mutations to enhance solution diversity, allowing the exploration of broader solution spaces and preventing stagnation (Figure 3). Figure 3 details the ACO's iterative solution construction, pheromone updating, and mutation processes, underscoring its efficacy in achieving faster convergence and higher optimization efficiency in complex industrial applications.

```
1. 1. Define Parameters and Variables
2.     - Stock lengths and costs
3.     - Piece lengths and required quantities
4.     - ACO parameters (number of ants, iterations, alpha, beta, decay rate, initial pheromone,
mutation base rate)
5.     - Initialize pheromone matrix
6.
7. 2. Define Heuristic Value Function
8.     - Calculate the utility of cutting a piece from a stock relative to the stock cost
9.
10. 3. Define Mutation Function
11.     - Randomly mutate solutions by repositioning pieces to potentially improve solution
diversity
12.
13. 4. Define Mutation Rate Adjustment Function
14.     - Adjust the mutation rate based on whether the new solution improves on the previous
best
15.
16. 5. Define Main ACO Solving Function
17.     - Initialize best solution and cost
18.     - For each iteration:
19.       - Generate solutions for all ants
20.       - Calculate fitness for each solution
21.       - Keep track of the best solution
22.       - Update pheromones based on solution quality
23.       - Mutate solutions
24.       - Adjust mutation rate
25.
26. 6. Define Solution Construction Function
27.     - Construct a solution by allocating pieces to stock based on a probabilistic decision
which considers both pheromones and heuristic value
28.
29. 7. Define Pheromone Update Function
30.     - Update pheromone levels based on the quality of solutions, promoting paths that lead to
better solutions
31.
32. 8. Define Fitness Calculation Function
33.     - Calculate the total cost of a solution based on the number and type of stocks used
34.
35. 9. Define Waste Calculation Function
36.     - Calculate the total waste for a solution as the difference between stock lengths and
the sum of piece lengths cut from them
37.
38. 10. Define Solution Printing Function
39.     - Print the best solution, its cost, waste, and computation time
40.
41. 11. Main Execution Block
42.     - Execute the ACO algorithm
43.     - Print the optimal cutting strategy and its details
44.
```

Figure 3 pseudo code for ACO with Dynamic Mutation [8]

3.2 Hybrid ACO with Local Search

The Ant Colony Optimization (ACO) framework, augmented with local search strategies, addresses the Cutting Stock Problem (CSP), an NP-hard combinatorial optimization challenge. This hybrid approach, drawing from Levine and Ducatelle's (2004) research on ACO applications in Bin Packing and CSP, combines probabilistic solution construction with strategic local searches to refine each solution, thus effectively minimizing waste and reducing stock costs. The algorithm's efficacy, documented through rigorous testing and analysis in a controlled environment, validates the superior performance and consistency of the hybrid ACO model, demonstrating its potential in complex industrial optimizations where traditional methods are inadequate.

```
1. 1. Import Libraries
2.     - Import numpy for numerical operations
3.     - Import random for generating random numbers
4.     - Import logging for tracking events and debugging
5.     - Import time to measure computation times
6.
7. 2. Define Problem
8.     - Specify stock lengths and their associated costs
9.     - Define lengths of pieces needed and their quantities
10.
11. 3. Set Parameters
12.     - Define the number of ants and iterations for the optimization
13.     - Set coefficients for pheromone influence (alpha), heuristic influence (beta), and
pheromone decay
14.     - Initialize a pheromone matrix to a constant value across all stock and piece
combinations
15.
16. 4. Define Heuristic Function
17.     - Calculate a heuristic value for cutting a specific piece from a specific stock,
adjusted by a random noise factor
18.
19. 5. Define Local Search Function
20.     - Apply local improvements to a solution by attempting to fit additional pieces into
existing stock cuts and rearranging pieces to minimize waste
21.
22. 6. Define Pheromone Update Function
23.     - Increase pheromone trails on paths that lead to better solutions, and apply a decay to
all pheromones
24.
25. 7. Define Solution Construction Function
26.     - Construct initial solutions based on pheromone trails and heuristic values, deciding
probabilistically which pieces to cut from which stocks
27.
28. 8. Define Fitness Calculation Function
29.     - Calculate the cost of a solution based on the number of stock pieces used
30.
31. 9. Define Waste Calculation Function
32.     - Calculate the total waste material from a solution by comparing the used length to the
total stock length
33.
34. 10. Define Solution Printing Function
35.     - Output the details of a solution, including total waste, computation time, cost, and
the specific cuts made from each stock type
36.
37. 11. Define Main Solution Function
```

```
38.     - Perform multiple iterations where each ant constructs a solution, local search is
applied, and pheromones are updated based on the quality of solutions
39.     - Track the best solution found across all iterations
40.
41. 12. Execution Block
42.     - Run the main solution function
43.     - Print the best solution and its details after all iterations
44.
```

Figure 4 pseudo code for ACO with Dynamic Mutation [8]

Full Source code for both experimentations can be found here [8]

4 Experimental Results

4.1 Performance Metrics

The performance of each proposed solution was evaluated based on three primary metrics: **the total cost of stock used, the extent of stock wastage**, and the **computational efficiency**, measured in time from initiation to solution. These metrics help assess both the effectiveness and efficiency of the algorithms under different problem complexities.

4.2 Results of Experiments

Table 1 illustrates a comparative analysis of the total costs incurred by ACO with Dynamic Mutation (Solution 1) and Hybrid ACO with Local Search (Solution 2) across the three problem instances. While both solutions approached the provided optimal costs, Solution 2 consistently yielded closer approximations, suggesting a more effective optimization strategy under varied conditions.

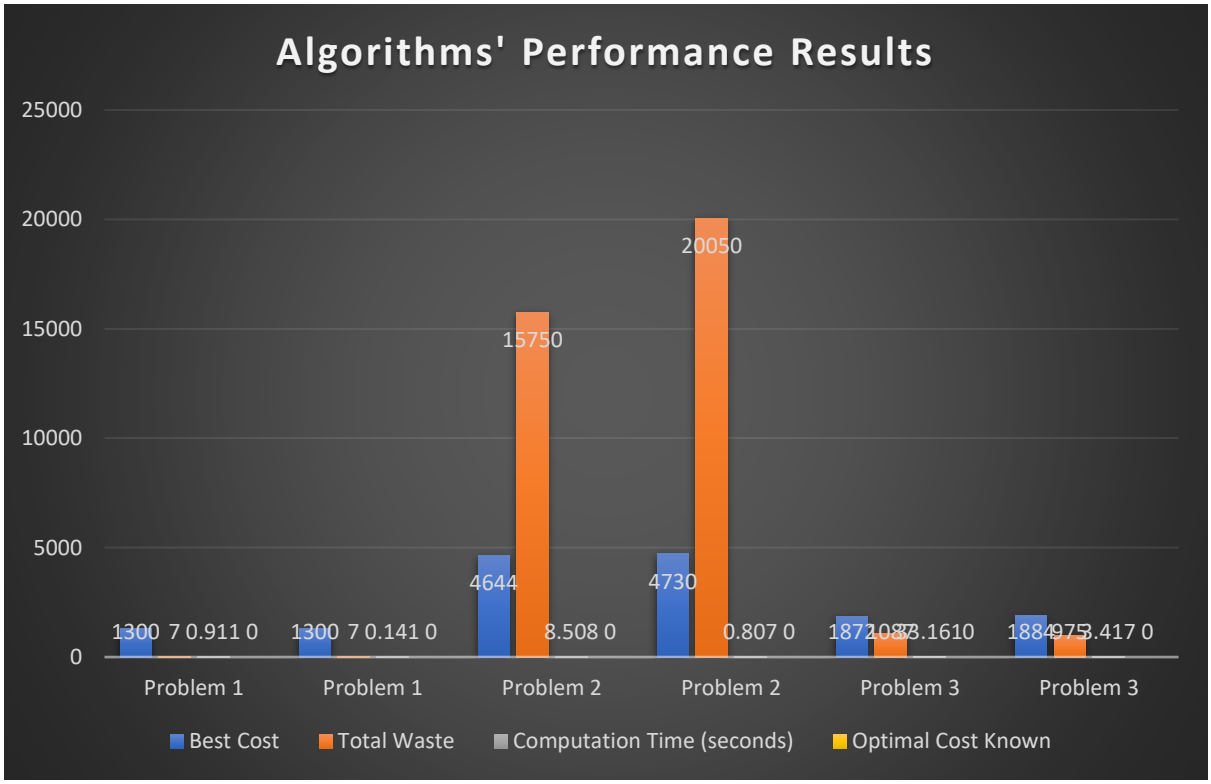
Problem	Algorithm	Best Cost	Total Waste	Computation Time (seconds)	Optimal Cost Known
Problem 1	ACO with Dynamic Mutation	1300	7	0.911	Yes (1240)
Problem 1	Hybrid ACO with Local Search	1300	7	0.141	Yes (1240)
Problem 2	ACO with Dynamic Mutation	4644	15750	8.508	Yes (3998)
Problem 2	Hybrid ACO with Local Search	4730	20050	0.807	Yes (3998)
Problem 3	ACO with Dynamic Mutation	1872	1087	33.161	No
Problem 3	Hybrid ACO with Local Search	1884	975	3.417	No

Figure 3 Algorithms Performance Results

5 Analysis and discussion

5.1 Comparative Analysis

In this study, two algorithms, ACO with Dynamic Mutation and Hybrid ACO with Local Search, were evaluated for their efficacy in solving a stock cutting problem across three scenarios of varying complexity. Key performance metrics included solution cost, waste minimization, and computation time. ACO with Dynamic Mutation often approached the optimal cost more closely, suggesting precise optimization, but with longer computation times, particularly in complex scenarios (e.g., 33.16 seconds for Problem 3). In contrast, Hybrid ACO with Local Search excelled in computation speed across all instances, offering a time-efficient solution despite sometimes higher costs and mixed results in waste minimization.



5.2 Conclusion of Experimental Results.

The experimental results highlight the strengths and limitations of each algorithm, suggesting that the choice between these two should be context-dependent:

- **Context of Efficiency**: Hybrid ACO with Local Search is favored in time-sensitive scenarios due to its rapid computation capabilities, appropriate for real-time applications despite potential compromises in cost and waste optimization.
- **Context of Cost and Precision:** ACO with Dynamic Mutation is preferred when minimizing costs is crucial and computation time is secondary, as its proximity to optimal cost can justify longer processing times in financially critical or high-cost material scenarios.
- **Waste Sensitivity:** The selection between algorithms for waste minimization varies with problem complexity; ACO with Dynamic Mutation may be better in complex situations, whereas Hybrid ACO with Local Search might be optimal in simpler contexts.

The study highlights the need to match algorithm selection with operational priorities like cost efficiency, time sensitivity, and waste reduction. Future research could focus on hybrid approaches and parameter adjustments to improve algorithm performance and extend testing to more diverse scenarios to better understand each algorithm's optimal conditions.

6 References:

1. Liang, K.-H., Yao, X., Newton, C., & Ho, D. (2002). A new evolutionary approach to cutting stock problems with and without contiguity. *Computers & Operations Research*, *29*, 1641–1659.
2. Lu, Q., Wang, Z., & Chen, M. (2008). *An Ant Colony Optimization Algorithm for the One-dimensional Cutting Stock Problem with Multiple Stock Lengths*. https://doi.org/10.1109/ICNC.2008.208
3. Levine, J., & Ducatelle, F. (2004). Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, *55*(7), 705–716. https://doi.org/10.1057/PALGRAVE.JORS.2601771
4. Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, *1*(4), 28–39. https://doi.org/10.1109/MCI.2006.329691

5. Reinertsen, H., & Vossen, T. W. M. (2009). The one-dimensional cutting stock problem with due dates. *European Journal of Operational Research*, *201*, 701–711. https://doi.org/10.1016/j.ejor.2009.03.042
6. Poldi, K. C., & Arenales, M. N. (2009). Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths. *Computers & Operations Research*, *36*, 2074–2081. https://doi.org/10.1016/j.cor.2008.07.001
7. Vishal Kadam, M., Jagdishprasad Jhabarmal, S., Vaze Shri Jagdishprasad Jhabarmal, V. M., & Todmal Shri Jagdishprasad Jhabarmal, S. R. (2021). *TACA: Trust Aware Clustering using ACO for Secure and Reliable Vehicular Ah hoc Network Routing*. https://doi.org/10.21203/rs.3.rs-710906/v1
8. Appiah, E. (no date) Emmanuelsnr1/Comp-Intelligence: Learnings for Computational Intelligence from MY MSC AI course at University., GitHub. Available at: https://github.com/EmmanuelSnr1/Comp-Intelligence (Accessed: 22 April 2024).

7 Appendix

7.1 Selected Piece Cuts from the 2 Solution

Algorithm	Stock Type	Stock Length	Pieces Cut
ACO with Dynamic Mutation	0	120	47, 51, 22
ACO with Dynamic Mutation	0	120	48, 63
ACO with Dynamic Mutation	0	120	67, 44
ACO with Dynamic Mutation	0	120	33, 39, 46
ACO with Dynamic Mutation	0	120	63, 35, 22
ACO with Dynamic Mutation	0	120	38, 57, 22
ACO with Dynamic Mutation	0	120	31, 63, 21
ACO with Dynamic Mutation	0	120	49, 42, 24
ACO with Dynamic Mutation	0	120	50, 63
ACO with Dynamic Mutation	0	120	47, 44, 29
ACO with Dynamic Mutation	0	120	35, 63, 21
ACO with Dynamic Mutation	0	120	56, 31, 32
ACO with Dynamic Mutation	0	120	35, 49, 33
ACO with Dynamic Mutation	0	120	38, 44, 38
ACO with Dynamic Mutation	0	120	33, 39, 44
ACO with Dynamic Mutation	0	120	33, 44, 33
ACO with Dynamic Mutation	0	120	44, 44, 21
Hybrid ACO with Local Search	0	120	57, 32, 31
Hybrid ACO with Local Search	0	120	22, 60, 34
Hybrid ACO with Local Search	0	120	51, 61

Hybrid ACO with Local Search	0	120	34, 24, 25, 32
Hybrid ACO with Local Search	0	120	31, 56, 24
Hybrid ACO with Local Search	0	120	56, 46
Hybrid ACO with Local Search	0	120	21, 22, 66
Hybrid ACO with Local Search	0	120	30, 35, 49
Hybrid ACO with Local Search	0	120	65, 32, 21
Hybrid ACO with Local Search	0	120	38, 38, 32
Hybrid ACO with Local Search	0	120	61, 22, 32
Hybrid ACO with Local Search	0	120	49, 63
Hybrid ACO with Local Search	0	120	35, 50, 31
Hybrid ACO with Local Search	0	120	35, 32, 46
Hybrid ACO with Local Search	0	120	57, 47
Hybrid ACO with Local Search	0	120	31, 56, 33
Hybrid ACO with Local Search	0	120	39, 51, 27