Internasional Journal of Integrative Sciences (IJIS)
Vol.2, No.7, 2023: 957-970

Genetic Algorithm Approach for Cutting Stock Problems in Construction Industries

Bambang Santoso[1*], Heri Haerudin[2]
Universitas Pamulang
Corresponding Author: Bambang Santoso dosen01692@unpam.ac.id

A R T I C L E I N F O

Keywords: Cutting Stock Problem, Genetic Algorithm, Reinforced Concrete, Constructions

Received : 5 May
Revised : 16 June
Accepted: 15 July

A B S T R A C T

Cutting Stock Problem (CSP) is a classic problem involving cutting long stocks into smaller products with certain quantities. The optimization is to find cutting patterns with minimum waste. In construction industries, CSP applies to steel bar cutting. The steel bar is an important element in making reinforced concrete. The length of the steel bars from the steel manufacturers is fixed, while the requirements for the constructions are varying. The problem is to find optimized way to cut long, fixed-length steel bars into smaller, varying length bars required in the constructions. The requirements are different from building to building, both in the lengths and quantities. Many studies have been extensively done on the subject, from Brute Force, Greedy Search to Linear Programming. In this paper the study focuses on Genetic Algorithm approach. The results look promising for Fitness Function 1 where the focus is to minimize waste. Waste ranges from 2.03% to 4.31%. Fitness function 2 and 3 do not emphasize merely on minimizing the waste, but also on contiguity. Therefore the residues are more, ranges from 2.21% to 4.91% for Fitness Function 2 and from 2.03% to 30.7% for Fitness Function 3

DOI: https://doi.org/10.55927/ijis.v2i7.4810
ISSN-E: 2985-9050
https://journal.formosapublisher.org/index.php/ijis

INTRODUCTION

Minimizing waste is the goal of Cutting Stock Problem (CSP). In many literatures (Abuhassan & Nasereddin, 2011) (Yan Chen, Xiang Song, Djamila Ouelhadj, 2017) (Pierini & Poldi, 2021), the example is in paper mills where production machines can produce fixed width paper rolls, but the requirements are for smaller-width rolls, although with the same length. The paper mill then needs to slice the paper rolls into several narrow-width rolls. CSP solution solves many real life problems in mass production industries such as steel, wood, glass, paper, textiles, and others (Yan Chen, Xiang Song, Djamila Ouelhadj, 2017) (Ogunranti & Oluleye, 2016). There are one-, two-, three-, and multi-dimensional CSP (Delorme, 2017). Several other variations like with and without contiguity gives more variety of CSP solutions.

When dealing with CSP, the large items are normally called Stock Materials or in short Stocks; the smaller items are called Products. The first problem is Assortment Problem to address the problem of how to find the optimal size of the Stocks. The second problem is Trim Loss Problem to find the way to efficiently cuts Stocks into Products in order to minimize the wastage. Both problems are known as Cutting Stock Problems (CSP). The case in this paper is Trim Loss Problem in one dimensional CSP.

In construction works, reinforced concrete is an important element of a building. It is the skeleton of a building that all other components rely on it. Reinforced concrete needs steel bars as its core with different diameters, lengths, and quantities.

The construction companies purchase steel bars from steel manufacturers or steel suppliers. The steel bars are of fixed length, while the requirements vary from one building to another. Example of the requirements for a building is as in Table 1 below.

Table 1. Example of Steel Bar Requirements

Diameter	Length	Quantity Required
10 mm	1.9 m	200
	2.2 m	150
	2.7 m	100
	3.1 m	200

The optimization problem then arises: how should they cut Stocks into Products to minimize the wastage? Additional question is: how many steel bars must the company purchase?

When talking about the significance of the problem, minimizing cost is always one of the goals to maximize profit with a condition that it does not sacrifice the quality of the finish product.

By planning ahead the cutting patterns, the company can buy at once all the steel stock requirements. This will avoid buying additional steel later when they find it is less than the requirements (will get less discount), or keep the Stocks in their stores when it is more than the requirements (need storage space).

Furthermore, as environment awareness becomes a more significant issue nowadays, the company want to preserve the environment. The steel wastage is bad pollutant. Minimizing steel wastage is good for environment.

LITERATURE REVIEW

The best way to find patterns for cutting Stocks is Brute Force method. With Brute Force, all possibilities are examined one by one and the optimal solution is found. This method is the best and is guaranteed to produce the optimal solution, but it takes a long time. The time required increases exponentially with the number of variables used. Because of this reason Brute Force is avoided unless the number of items is low.

Greedy Search is the second approach, which is looking for optimal solutions at each level (Feo & Resende, 1995) (Santoso et al., 2019). This approach results in a fast algorithm. But sometimes it stops at the pseudo peak so that optimum results are questionable. (Lazar & Zuazua, 2022)

Several other approaches are also carried out such as Linear Programming (Abuhassan & Nasereddin, 2011) (Porumbel, 2022), Simulated Annealing (Kokten & Sel, 2022).

Another approach that will be used in this study is the Genetic Algorithm (GA). The GA approach uses chromosomes as candidate solutions, imitating living things.

METHODOLOGY

Formal Definition of the Problem

The problem in this study is to find optimal or near-optimal solution for single length stocks 1-dimensional CSP without contiguity. For illustration purposes, the requirements in Table 1 will be used through out the paper.

Suppose:

l_i = the length of i-th requested item

c_i = quantity required for i-th requested item

The solution to fulfil is the following.

$c_1 l_1 + c_2 l_2 + \ldots + c_n l_n, \quad i = 1, 2, \ldots, n$

In the example in Table 1 this translates to fulfil the following.

200 (1.9) + 150 (2.2) + 100 (2.7) + 200 (3.1),

$i = 1, 2, 3, 4$

The total length of the required stocks is:

$z = k * y$

k = number of stocks required

y = stock length

In the example in **Error! Reference source not found.**, the total length will be:

$z = k * 12 = 12k$

Cutting patterns for each stock can be listed as:

$p_{11} l_1 + p_{12} l_2 + \ldots + p_{1n} l_n \leq y$

or

$$\sum^{n} p_{ji} l_i \leq y$$

$i=1$

p_{ji} = number of i-th item in j-th pattern

i = 1, 2, …, n

j = 1, 2, …, m

For example, three cutting patterns for 12 meter stocks are listed below.

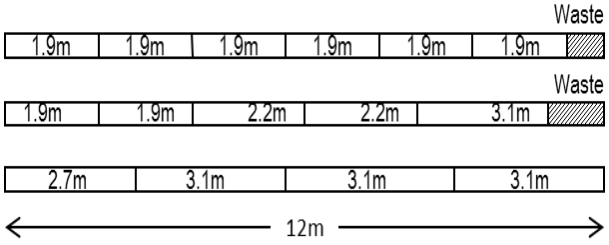

Figure 1. Example of Cutting Patterns to Stocks

The patterns above can be written as follows.

P1=6(1.9)+0(2.2)+0(2.7)+0(3.1)

P2=2(1.9)+2(2.2)+0(2.7)+1(3.1)

P3=0(1.9)+0(2.2)+1(2.7)+3(3.1)

For Pattern1 (P1), the sum of the cuts is 11.6m. Thus the waste is 0.4m. Pattern2 (P2) has waste of 0.7m, and Pattern3 (P3) does not create any waste.

It is easily understood that the total length of any patterns must be less than or equal to the stock length, in this case is 12 meters.

We can list all patterns in the following table.

$p_{11}l_1 + p_{12}l_2 + \ldots + p_{1n}l_n$

$p_{21}l_1 + p_{22}l_2 + \ldots + p_{2n}l_n$

.

.

$p_{m1}l_1 + p_{m2}l_2 + \ldots + p_{mn}l_n$

For the example of requirements in Table 1, counts of all combinations is as follows. (Kenneth H. Rosen, 2007)

`C(p,m)=p!/(m!*(p-m)!)`

with p! = 1*2*3*...*p

For p=9 and m=3 the calculation will give us 84 combinations. However, not all combinations are valid. After scrutinizing them, the following is the list of all valid patterns.

Table 2. List of All Possible Patterns in the Example

Pattern	Quantity	l_1 = 1.9 m	l_2 = 2.2m	l_3 = 2.7m	l_4 = 3.1m	Waste
1	$x1$	6	0	0	0	0.6
2	$x2$	5	1	0	0	0.3
3	x*3*	4	2	0	0	0
4	x*4*	4	0	1	0	1.7
5	x*5*	4	0	0	1	1.3
6	x*6*	3	1	1	0	1.4
7	x*7*	3	1	0	1	1
8	x*8*	3	0	2	0	0.9
9	$x9$	3	0	1	1	0.5
10	x*10*	3	0	0	2	0.1

11	x*11*	2	3	0	0	1.6
12	x*12*	2	2	1	0	1.1
13	x*13*	2	2	0	1	0.7
14	x*14*	2	1	2	0	0.6
15	x*15*	2	1	1	1	0.2
16	x*16*	2	0	3	0	0.1
17	*x17*	1	4	0	0	1.3
18	*x*18	1	3	1	0	0.8
19	*x19*	1	3	0	1	0.4
20	*x20*	1	2	2	0	0.3
21	*x21*	1	1	0	2	1.7
22	*x22*	1	0	2	1	1.6
23	*x23*	1	0	1	2	1.2
24	*x24*	1	0	0	3	0.8
25	*x25*	0	5	0	0	1
26	*x26*	0	4	1	0	0.5
27	*x27*	0	4	0	1	0.1
28	*x28*	0	3	2	0	0
29	*x29*	0	2	1	1	1.8
30	*x30*	0	2	0	2	1.4
31	*x31*	0	1	3	0	1.7
32	*x32*	0	1	2	1	1.3
33	*x33*	0	1	1	2	0.9
34	*x34*	0	1	0	3	0.5
35	*x35*	0	0	4	0	1.2
36	*x36*	0	0	3	1	0.8
37	*x37*	0	0	2	2	0.4
38	*x38*	0	0	1	3	0

There are 38 possible patterns in the above example. We can use each pattern multiple times to fulfil the requirements. The quantity x_j is the number of how many times a specified pattern is used in the solution, j = 1, 2, …, 38. Total length of stocks to cut is:

$ky = y\,(x_1 + x_2 + \ldots + x_m)$

k = number of stocks to cut

y = lengths of the stock

$12k = 12\,(x_1 + x_2 + \ldots + x_m)$

Or, in general,

$z = y\,(x_1 + x_2 + \ldots + x_m)$

Without effecting the optimization, we can write the problem as minimizing the number of stocks to cut.

$$\min Z = x_1 + x_2 + \ldots + x_m$$

Or, alternatively, the problem can be stated as to minimize the wastage produced by the cut.

$$\min W = \sum_{m} w_j x_j$$

$j=1$

Constraints:

$\sum_{j=1}^{m} a_{ij}x_j = \mathcal{N}i$ for $i = 1, 2, \ldots, n$

$a_{ij} \geq 0$, $x_j \geq 0$,

a_{ij}, x_j are integers

$j = 1, 2, \ldots, m$

where:

n = number of requirement types

m = number of all possible patterns

w_j = waste of pattern j

a_{ij} = number of pieces of item i in pattern j

x_j = number of pattern j used in the solution

$\mathcal{N}_i$ = number requirements of item i

The approach on this study emphasizes on creating all possible patterns as the search space, and uses multiple numbers of the patterns as the base of optimization. Duplications are allowed in a search space. The way to cut a stock into products is called a pattern.

Encoding

Encoding is a part of GA that is tailored to the problem. Any implementer of GA shall find the way to represent the problem that GA can understand and thereafter can help to solve (Mitchell, 1999). In this study, the encoding is as follows.

All the stocks to cut are expressed in z.

$z = x_1 + x_2 + \ldots + x_m$

subject to:

$\sum_{j=1}^{m} x_{ij} \geq N_i$

where:

z = number of stocks to cut

x_j = number of stocks to cut with pattern j

x_{ij} = number of i-th product in j-th pattern

N_i = quantity required for i-th product

i = 1, 2, …, n

j = 1, 2, …, m

Other trivial requirements

$x_{ij} \geq 0$, integer

$N_i \geq 0$, integer

The study uses chromosome encoding as follows

$[x_1, x_2, \ldots, x_m]$

with x_j is defined as above, j = 1, 2, …, m.

Initial Population

Initial population is as generation 0 of the iteration in GA. The task includes choosing initial patterns, and assigning random quantities for those patterns. The

process is repeated until a certain number of valid solutions are generated. Detail process is in the following.

For each stock, a random number between 1 to m is generated to choose the pattern to use. After getting the pattern, the function uses random number again to assign quantity for this pattern.

For example, for 38 patterns in Table 2 a random number gives 3 for the pattern and 14 for the quantity. The chromosome will be:

[0-0-14-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0]

For the next pattern, again another 2 random numbers are required, one for the pattern and one for the quantity. After a certain number of patterns are assigned, the chromosome is ready as one individual in the population. Example of a chromosome is as below.

[0-0-14-0-0-0-0-36-0-0-0-0-0-0-0-0-0-3-0-0-0-0-0-0-0-0-0-1-0-0-0-0-30-0-0-0-0-0]

Validator

Chromosomes in initial population are not necessary optimal, but must be valid solutions for the problem. The initial chromosomes as created above may not be valid ones. To be valid, it must satisfy the following.

$$\sum_{j=1}^{m} x_{ij} \geq N_i \qquad i=1, 2, \ldots, n$$

For this, the algorithm needs to build a validator. The function of a validator is to make sure that the chromosome is valid to be a candidate solution. Moreover, the validator in this case acts also as an optimizer to some extent. The validator has two sub functions: Adder and Reducer.

Adder will check whether all product requirements are satisfied by the chromosome. If not, it will take patterns randomly to add the difference until all required items fulfilled. Then, the reducer searches a pattern, and checks whether reducing the pattern still make it a valid chromosome. If it is the case, the reducer will subtract some amount from the quantity of the selected gene.

After all chromosomes are validated, it is time now to calculate the fitness of each chromosome using fitness function.

Fitness Function and Roulette Wheel

To know which individuals are strong, GA needs to implement a fitness function. The function will ascertain each chromosome in the population and attach a fitness index to them (Kumar, 2015). The fitness shall reflect the aim of the solution; in this case is to minimize the wastage. Again, GA leaves the definition of the fitness function to the implementer to choose or create his own.

In this study, the following fitness function is used.

fitness = 1 – cost

The cost is defined as one of the three functions below.

a) Cost Function 1

The cost function to measure the wastage is as below.

$$\text{cost} = \frac{1}{\text{m}} \sum_{j=1}^{m} \frac{w_j}{L_j}$$

where:

w_j = wastage of j-th pattern

L_j = length of the stock of j-th pattern

j = 1, 2, …, m

m = number of patterns

The above cost function is for multiple stock lengths. For single stock length, all L_j are the same for all patterns.

b) Cost Function 2

$$\text{cost} = \left(\frac{1}{\text{m} + 1}\right)\left(\sum_{j=1}^{m} \frac{w_j}{L_j} + \sum_{j=1}^{m} \frac{\text{V}_\text{j}}{L_j}\right)$$

where:

V_j = number of stocks with wastage

The first term is to calculate wastage as the whole, as this is the aim of the algorithm. The second term is to put emphasis on minimizing number of stocks with wastage. Less stocks with wastage is more desirable as it means the wastage can be concentrated on a few stocks. For instance, from 100 stocks used, 5 stocks with 1m wastage is better than 10 stocks with 0.5m wastage each, although the number of the stocks and the total wastage are the same.

c) Cost Function 3

In cost function 3, the second term is multiplied by one third of m, to make the following cost function.

$$\text{cost} = \left(\frac{1}{m + (m/3)}\right)\left(\sum_{j=1}^{m} \frac{w_j}{L_j} + \frac{m}{3}\sum_{j=1}^{m} \frac{V_j}{L_j}\right)$$

All notations have the same meaning as the above. With this, the emphasis on the second term is always proportionate (that is, one third) to the number of genes in one chromosome.

Each chromosome has fitness that ranges between 0 (the weakest) and 1 (the strongest). To normalize this as a roulette wheel, the algorithm will sum up all the values and then divide each with the sum. This will linearize the fitness to be proportional relative to other chromosomes. Next, the algorithm will put this proportion one on top of the other that the last chromosome will have a value of 1.

With the help of roulette wheel as above, we are ready to implement GA operators.

GA Operators

This study uses three GA operators to build new populations; namely crossover, mutation, and elitism.

a) Crossover

Most literatures discuss the crossover as one-split and two-split types (Mitchell, 1999)(Saxena, 2016)(Amjad et al., 2018). The process of one-split crossover is as follows. Pick 2 chromosomes, split at one common point. Take first part of the chromosome from first parent, and take the second part from the second parent.

Process of two-split crossover is the same, only that the common points are two instead of one. Pick the first part from the first parent, the second part from the second parent, and the third part is again from the first parent.

In this study we use Uniform Order Crossover (UOCX). It works by copying all the genes from parent1 to the child. A random number afterward is generated between 1 and m. The gene from parent2 at this random position is then copied to the child.

This crossover may drastically take the child far away from the parents. To reduce the effect of crossover, Half UOCX is also implemented.

The process of Half UOCX is similar to that of UOCX with slight modifications. Instead of copying the gene from the parent2, the gene is replaced by the median of the two parents.

b) Mutation

Mutation requires only one parent (Mitchell, 1999)(Amjad et al., 2018). It changes several genes to become a new individual. The process is as follows: take randomly a gene, and add or subtract an amount to it. Repeat for a certain number of genes. Two random numbers are required here.

1. The first is to choose the genes.
2. The second random number is to get an amount to add or to subtract from the gene. This amount should be small enough that it does not destroy the optimality. The default rate is 0.005 from the total quantity requirements.

For example in Table 2 the total requirements is 150+120+200+100=570. The limit amount for 0.005 is 2 (integer, rounded down). The random number generated is therefore between -2 and +2.

This amount is added to or subtracted from the quantity of the specified gene. If the result is negative, the quantity is set to zero.

The algorithm repeats the process until a certain number of genes are modified. After the process, the validator shall be called later to make sure the solution is valid.

c) Elitism

Elitism is preserving chromosomes to the next generation (Mitchell, 1999). These chromosomes are copied to the next generation without any changes.

d) Combination of Genetic Operators

In this study several ways are implemented to combine the three genetic operators.

Crossover Only

New generation is constructed by means of crossover operator. Mutation is not used. Elitism is used when crossover rate is less than 1.

Mutation Only

Only mutation operator is used to construct new generation. Crossover is not used. Elitism is used when the mutation rate is less than 1.

Separate Crossover and Mutation

Part of the new generation is constructed by crossover, the other part are by mutation.

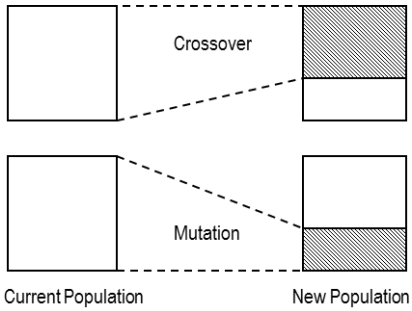

Figure 2. Separate Crossover and Mutation, Part of the New Population is Generated by Crossover, and Other Part is by Mutation

Crossover Then Mutation

The process creates intermediate population as a product of crossover and then applies mutation to this intermediate population. Mutation is not necessarily applied to 100% of the intermediate population. Part mutation is permissible.

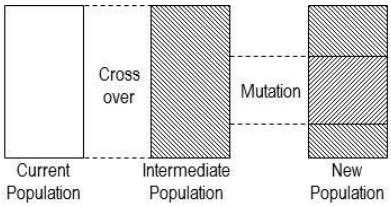

Figure 3. Crossover Then Mutation, an Intermediate Population is Generated Entirely by Crossover, Mutation is then Operated on Intermediate Population

Iteration

The full cycle of the GA process is as below.

1. Get Parameters and Requirements.
   Parameters and requirements are stored in files.
2. Create Search Space.
3. Create Initial Population.
4. Running Validator.
   Check whether the population fulfills the requirements.
5. Calculate the Fitness.
6. Check to Stop.
   Process will stop when the fitness value is matching the requirement, or the iteration is exceeding the maximum allowed.
7. Create New Generation.
8. Go to step (4).

RESULTS

The last task in the study is to run the programs on test-case data and observe the results. The data requirement files are taken as disperse as possible. There are 10 data requirements files as the test cases.

The parameter files are tweaked first to get the best result. Below are the parameters used to run the program.

```
crossoverRate=1
crossoverItemRate=0.1
crossoverHalf=no
mutationRate=0.1
mutationItemRate=0.05
mutationQtyRate=0.07
crossoverThenMutate=yes
population=100
initPatternRate=0.3
maxIteration=1000
targetFitness=1
```

Figure 4. Parameter file

The program are executed 20 times to get the best result. Each run may give different results as the random numbers are always different. The best results are as follows.

Table 3. Test Results for Fitness Function 1

Run	Waste (m)	Stock used (pcs)	Waste (%)	Stock with waste (pcs)	Run time (s)
req01	29	119	2.03	83	1.0
req02	275	1083	2.12	482	1.4
req03	41	120	2.63	71	2.7
req04	358	1029	2.68	679	4.7
req05	73	178	2.93	158	6.5
req06	541	1290	3.00	1212	11.6
req07	117	212	3.94	206	13.6
req08	837	1560	3.58	1521	23.8
req09	217	315	4.31	224	74.9
req10	1405	2788	3.15	1923	78.7

Table 4. Est Results for Fitness Function 2

Run	Waste (m)	Stocks used (pcs)	Waste (%)	Stocks with waste (pcs)	Run time (s)
req01	41	120	2.85	72	1.0
req02	287	1084	2.21	632	1.5
req03	41	120	2.63	78	2.9
req04	371	1030	2.77	733	4.8
req05	73	178	2.93	137	6.5
req06	597	1294	3.30	1070	11.3
req07	117	212	3.68	200	13.5
req08	837	1560	3.58	1423	23.9
req09	249	317	4.91	225	76.7
req10	1261	2779	2.84	1915	76.5

Table 5. Test Results for Fitness Function 3

Run	Waste (m)	Stocks used (pcs)	Waste (%)	Stocks with waste (pcs)	Run time (s)
req01	65	122	4.44	56	1.0
req02	263	1082	2.03	269	1.4
req03	145	128	8.71	49	2.7
req04	553	1044	4.07	469	4.8
req05	507	209	17.3	135	6.5
req06	933	1318	5.06	1002	11.5
req07	462	235	13.1	189	14.7
req08	1107	1578	4.68	1327	24.8
req09	2137	435	30.7	341	71.7
req10	13853	3566	24.3	2531	119.5

DISCUSSION

The following is the observations from running the tests.

1. There are many fine tunes of parameters should be done to GA program. Running GA with non-optimal parameters may yield bad results. However, optimizing parameters in GA is a tricky task as changes in one parameter may affect the balance of other parameters. Some wrapper program might help in determining best parameters for GA program.
2. Sufficient number of iterations and also several runs of the program are needed to get satisfactorily results from GA program. Run GA program several times will get different results as the program uses random numbers to generate the results. The implementer should run the program multiple times and grab the fittest result.
3. Fitness function is an interesting feature of GA. One can change the aim only by changing fitness function. In this study three fitness functions are implemented. Other fitness functions can be incorporated quickly without changing much code, provided that the encoding is the same.
4. The performance of GA varies according to the number of patterns found. The more patterns it creates, the slower the performance is.
5. Although basic steps in GA are quite established, the implementations vary considerably from problem to problem. This makes GA difficult to take off, as general solution cannot be built. This is a challenge for GA practitioners.
6. The percentage of the waste is quite small especially for fitness function 1, ranging from 2.03% to 4.31%. Other fitness functions give different results as the aim is not merely the waste.
7. Fitness function 3 has strange behaviour in this study. The aim of fitness 3 is to get minimum number of stocks with waste. The numbers are correct for 8 requirements. However, for the requirement 9 and 10 the numbers are quite reversing. In these two requirements, fitness 3 has more stocks with waste compared to fitness 1 and fitness 2. It needs more study to find out the cause.

8. Overall the program can give the solution requested: (1) the patterns to cut Stocks into Products, and (2) the number of steel bars the company has to purchase to fulfill the requirements.

FURTHER STUDY

Further works on this topic are envisaged as below.

1. Mutation turns out to be the dispersing function in the GA and crossover is the converging function. One may wish to reverse the order, mutation first and then crossover.
2. Better encoding should be implemented, as the running time is slow when a lot of patterns are created.
3. Wrapper program can be implemented before GA program to select best parameters to use.

REFERENCES

Abuhassan, I. A. O., & Nasereddin, H. H. O. (2011). Cutting Stock Problem : Solution Behaviors. *International Journal of Recent Research and Applied Studies*, *6*(March), 429–433. https://www.researchgate.net/publication/281120697_CUTTING_STOCK_PROBLEM_SOLUTION_BEHAVIORS

Amjad, M. K., Butt, S. I., Kousar, R., Ahmad, R., Agha, M. H., Faping, Z., Anjum, N., & Asgher, U. (2018). Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems. *Mathematical Problems in Engineering*, *2018*. https://doi.org/10.1155/2018/9270802

Delorme, M. (2017). Mathematical models and decomposition algorithms for cutting and packing problems. *Dottorato Di Ricerca in Automatica e Ricerca Operativa Ciclo*. https://doi.org/10.1007/s10288-017-0365-z

Feo, T. A., & Resende, M. G. C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, *6*(2), 109–133. https://doi.org/10.1007/BF01096763

Kenneth H. Rosen. (2007). Discrete Mathematics and Its Applications. In *Mc Graw Hill* (6th ed.). Mc Graw Hill. https://doi.org/10.13109/9783666538452.10

Kokten, E. S., & Sel, Ç. (2022). A cutting stock problem in the wood products industry: a two-stage solution approach. *International Transactions in Operational Research*, *29*(2), 879–907. https://doi.org/10.1111/itor.12802

Kumar, D. (2015). Comparative Study of Genetic Algorithm Performed in a Single Generation for two Different Fitness Functions Technique f(x) = x^2 and f(x) = x^2+1. *International Journal of Computer Applications*, *128*(17), 7–15. https://doi.org/10.5120/ijca2015906572

Lazar, M., & Zuazua, E. (2022). *Greedy Search Of Optimal Approximate Solutions*.

Mitchell, M. (1999). An Introduction to Genetic Algorithms. In *A Bradford Book The MIT Press*. A Bradford Book The MIT Press. https://doi.org/10.1162/artl.1997.3.63

Ogunranti, G. A., & Oluleye, A. E. (2016). Minimizing waste (off-cuts) using cutting stock model: The case of one dimensional cutting stock problem in wood working industry. *Journal of Industrial Engineering and Management*,

9(3), 834–859. https://doi.org/10.3926/jiem.1653

Pierini, L. M., & Poldi, K. C. (2021). Lot sizing and cutting stock problems in a paper production process. *Pesquisa Operacional*, *41*(Special issue). https://doi.org/10.1590/0101-7438.2021.041s1.00235094

Porumbel, D. (2022). Projective Cutting-Planes for Robust Linear Programming and Cutting Stock Problems. *INFORMS Journal on Computing*, *34*(5), 2736–2753. https://doi.org/10.1287/ijoc.2022.1160

Santoso, B., Prasetiyo, S. M., & Wijoyo, A. (2019). Meminimalkan Sisa Pemotongan Besi Beton dalam Proyek Konstruksi. *Jurnal Informatika Universitas Pamulang*, *4*(2), 73. https://doi.org/10.32493/informatika.v4i2.3204

Saxena, A. (2016). Review of Crossover Techniques for Genetic Algorithms. *International Journal of Trend in Research and Development*, *3*(5), 347–349.

Yan Chen, Xiang Song, Djamila Ouelhadj, Y. C. (2017). A heuristic for the skiving and cutting stock problem in paper and plastic film industries. *International Transactions in Operational Research*, *26*(1), 157–179.