A Cuckoo Search Algorithm for 2D-cutting Problem in Decorative Ceramic Production Lines with Defects

Javier Monzon[1], Rony Cueva[1], Manuel Tupia[1] and Mariuxi Bruzza[2]
[1]*Department of Engineering, Pontificia Universidad Catolica del Peru, Av. Universitaria 1801 San Miguel, Lima, Peru*
[2]*Faculty of Tourism and Hospitality, Universidad Laica Eloy Alfaro de Manabi, Manta, Ecuador*

Keywords: Cuckoo Search Algorithm, 2D-Cutting Problem, Optimization, Bio-Inspired Algorithms.

Abstract: The residues generated from ceramic cuttings are one of the major causes of waste in the ceramic production industry, with losses being around 40% of the used material. Hence, reducing residues from materials used is critical for lowering the production cost. It is worth mentioning that in this industry the material also shows high rates of defects, a constraint which most researches dealing with the 2D-cutting problem lack. This paper develops a bio-inspired metaheuristic called Cuckoo Search to solve the problem of exposed material cutting as an alternative solution to the genetic algorithm already developed by the authors, which will also be used to measure the Cuckoo Search algorithm performance.

1 INTRODUCTION

Ceramic listels are generally rectangular-shaped decorative products obtained from larger ceramic cuttings. These laths can be previously decorated or can be decorated after the cutting. They are used in any home setting such as floors, bathrooms, kitchens, so that their use and production is largely widespread in the sector. Due to the components' properties in ceramic products, cracks may appear in these products which may result in damages to the material overtime and because of environmental conditions such as temperature fluctuations (Pastor, 2002). Likewise, these materials show a fragile breakage degree so that this type of failures by breaking is common.

For the manufacture of ceramic listellos, cutting ceramic pieces of different sizes (40x40, 45x45, 60x60, etc.) is necessary. Hence, determining the manner of obtaining listellos through the cutting of one ceramic piece of larger size is pivotal, in such a way that the loss resulting from the cutting is minimized and thus total demand is met (Talbi, 2009) (Tupia, Cueva and Guanira, 2013), (Tupia, Cueva and Guanira, 2017).

The emergence of variations of this problem – many of them very similar– urged Dyckhoff to put forward a typology to classify cutting and packaging issues (Dyckhoff, 1990). Based on said research, an improved typology is proposed in (Wäscher, Haußner and Schumann, 2007) to identify some shortcomings and deal with the variations required in our research. The characteristics of ceramics to be considered are as follow:

- 1) Dimensionality: Sizes established for the production line.
- 2) Type of assignment: Maximization, minimization.
- 3) Range of small objects: Identical, slightly heterogeneous, strongly heterogeneous.
- 4) Range of large objects: Identical, slightly heterogeneous, strongly heterogeneous.
- 5) Shape of final small items: Rectangles, circles, boxes, cylinders

Additionally, other characteristics were proposed to the problem regarding orientation and cutting patterns (Lodi, Martello and Vigo, 1999), (Du and Swamy, 2016):

- 6) Orientation: Items may require a fixed orientation or may be rotated 90°.
- 7) Guillotine cuttings: Cuttings may either require or not end-to-end cuttings in parallel to both sides: that is to say, if this is done in a rectangular piece, only two rectangles would be obtained.

Proposed constraints to be considered within the problem in this research are as follows (Booth, 2017), (Arbib et al, 2018):

Monzon, J., Cueva, R., Tupia, M. and Bruzza, M.
A Cuckoo Search Algorithm for 2D-cutting Problem in Decorative Ceramic Production Lines with Defects.
DOI: 10.5220/0007346705470553
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 547-553
ISBN: 978-989-758-350-6

- RF: Pieces may rotate 90º (R) and not all cuttings are required to be made with guillotine (F).
- RG: Pieces may rotate 90º (R) and all cuttings are required to be made with guillotine (G).
- OF: Direction of pieces is fixed (O) and not all cuttings are required to be made with guillotine (F).
- OG: Direction of pieces is fixed (O) and all cuttings are required to be made with guillotine (G)

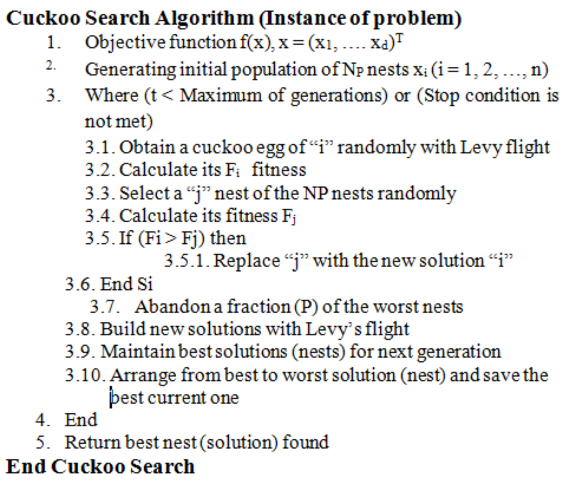
2 CUCKOO SEARCH ALGORITHM

This is a search metaheuristic algorithm for optimization problems. It can be classified as a "swarm intelligence algorithm" since this is inspired by the cuckoo bird's collective reproductive behavior in which some species usually lay their eggs in other birds' nests. If one of the host birds discovers an alien egg, the bird can either throw said egg away or abandon the nest and build another one elsewhere. However, many of these eggs are very similar to the hosts' eggs, and as they are placed in several nests the likelihood of being abandoned is reduced (Yang and Deb, 2009).

Those nests that are "discovered" by the host birds are left and removed from the population and replaced with new nests (with new random solutions). The following idealized rules are taken into account (Yang and Deb, 2009):

- Each cuckoo may lay one single egg at a time and place it in a nest (solution) selected randomly.
- Nests sheltering better quality eggs (solutions) will be transferred giving birth to new generations of cuckoo birds.
- The host bird may detect a cuckoo egg (that doesn't belong) with a probability P ∊ [0, 1]. The bird may either throw the cuckoo's egg away from the nest or abandon the nest (put it simply, this is equal to replacing P ratio of the worst nests)

Below is the pseudo code of a Cuckoo Search algorithm in the version to be implemented in our research (Iglesias et al, 2018):



```
Cuckoo Search Algorithm (Instance of problem)
  1. Objective function f(x), x = (x1, .... xd)^T
  2. Generating initial population of NP nests xi (i = 1, 2, ..., n)
  3. Where (t < Maximum of generations) or (Stop condition is not met)
        3.1. Obtain a cuckoo egg of "i" randomly with Levy flight
        3.2. Calculate its Fi fitness
        3.3. Select a "j" nest of the NP nests randomly
        3.4. Calculate its fitness Fj
        3.5. If (Fi > Fj) then
              3.5.1. Replace "j" with the new solution "i"
     3.6. End Si
        3.7. Abandon a fraction (P) of the worst nests
     3.8. Build new solutions with Levy's flight
     3.9. Maintain best solutions (nests) for next generation
     3.10. Arrange from best to worst solution (nest) and save the
           best current one
  4. End
  5. Return best nest (solution) found
End Cuckoo Search
```

Figure 1: Cuckoo search algorithm general approach.

3 CONTRIBUTION

3.1 Objective Function

To minimize the residue generated from cutting material as the primary objective, the percentage of material considered as a defect must be minimized. For this article, the stock region that was not used is regarded as residue as well as the pieces located on the region with defects. It is worth mentioning that the objective function to be put forward favours pieces that are not found on faulty regions and, but if they are, the pieces would be smaller, as they would generate more residue otherwise. This statement can be formalized in the following objective function (called FO):

$$\mathrm{F.O.} = \mathrm{Mín}\left(1 - \frac{\sum_i^N \mathrm{areaPieces}_{\mathrm{stock}_i}}{\sum_i^N \mathrm{area}_{\mathrm{stock}_i}}\right) \times 100 \qquad (1)$$

In the equation (1):

- N: The number of stocks used to fulfill the order. The minimum value to be taken is 1 (all the order is fulfilled with one stock) and the maximum value is equal to the quantity of pieces requested (one piece per stock).
- Stock$_i$: This is the i-nth stock with a width, height related to a list of defects represented by defined pieces.
- areaPieces_(stock_i): This is the addition of the areas of all pieces in each stock_i, that would be equal to all the pieces of the order. This is

calculated internally by multiplying width and height of each piece.

- area_(stock_i): This is the addition of the area occupied by each stock_1. It is calculated internally by multiplying the width and height of each stock.

3.2 Constrains

The number of available stocks must be higher or equal to the number of pieces requested: Constraint indicating that the quantity of available stocks must be sufficiently large to cover any solution to the problem in the best and worst scenarios. Likewise, the quantity of pieces must be 1 as a minimum to be considered an order.

The rectangle covering the pieces must cover the piece to be cut: Constraint indicating that there is a rectangle capable of covering each of the pieces (not sticking out of said rectangle). It is preferred that such rectangle is minimum as at first the stock is considered as the union of all possible stocks (regarded as infinitum) both in width and height before dividing them into multiple stock.

Pieces cannot overlap: The constraint sought is that no overlapping of pieces in the arrangement occurs. Pieces must not cover regions with defects. The purpose is that no overlapping occurs of pieces requested with the regions with defects, represented as pieces.

3.3 Data Structures

To represent data structures in the algorithm proposed, the following variables will be used:

3.3.1 List Solution

The structure will allow representing a solution considering a large stock, which will consist of a chain of numeric characters and letters for which a list of chains has been used for convenience so they can support numbers, letters and empty chains. Then an example of the solution to the problem is introduced, which has a N size (occupied by pieces and operators).

Table 1: Example of a solution to the problem.

Position	0	1	2	3	4	5	6
Value	1R	5N	H	7N	6N	V	2N

where the value of the list, depending on its type, represents the following:

- Numeric type value: Represent the identifier of a piece and its orientation, that is to say the value "1R" refers to piece 1 with rotation of 90° while the value "5N" refers to piece 5 without rotation of 90°.
- Character type value: Represent one operator, represented by characters H(horizontal) or V(vertical).

3.3.2 Class Rectangle

This class will allow to define a piece or set of pieces (block) from the two coordinates x, y which are interpreted with respect to the origin (0, 0) and width and height represented by variables w and h. Likewise there is a rotation variable, which will indicate if the piece is rotated 90° with respect to that originally recorded.

3.3.3 List Pieces

This structure will allow to represent the demand of pieces, which will be indicated by a list of pieces (using the rectangle class). An example of a list of pieces is shown in the table below:

Table 2: Example of List of Pieces.

Position	id	X	Y	W	H
0	1	0	0	40	8
1	5	0	0	40	25

Where variables represent the following:

- Id: The identifier of pieces: [piece_1, piece_2,… piece_N].
- x, y: Coordinates of pieces with respect to the origin (0, 0). Initially all of them will occupy the position (0, 0).
- w, h: The width and height of pieces

3.3.4 Class Stock

This stock class will allow to represent stock material which has a width and height with the peculiarity that unlike any piece, the stock is related to a list of defects which will be indicated by a list of pieces. Additionally, prior to the fitness calculation, an algorithm is run in which the solution is divided into multiple stocks so that each stock will be related to a list of pieces, initially empty.

3.3.5 List of Stocks

This structure will allow to represent the available stock material expressed as a list of stocks. Then, an example of a list of N stocks available is introduced with each showing a maximum of M defects:

Table 3: Example of List of Stocks.

Position	id	X	Y	$Defect_1$	$Defect_2$
0	1	10	20	$Piece_1$	$Piece_2$
1	2	30	50	$Piece_1$	$Piece_2$

Where variables represent the following:

- Id: The identifier of the stock: [stock_1,stock_2,… stock_N].
- w, h: The width and height of the stock.
- $Defect_i$: It represents a region with defects, which will be represented by the same structure than a piece (see Rectangle class in previous section), indicating the position relative to the stock they belong to.

3.3.6 Class Node

This class will allow to define a binary tree in which post-fixed notation may be represented in the algorithm representation. It has two attributes of the same kind that represent left and right children. Besides, it has a rectangle variable, which will indicate the occupied area and the position of a piece (if this is a leaf) or a block (if this is the union of several pieces). The latter will be defined through the integration type variable, which may be "H" o "V".

3.4 Integration Operators

The representation of the solution in a problem influences on the complexity and implementation of the solution. For this work, a post-fixed representation will be used through a list of characters, as proposed in (Leon et al, 2007). To define such structure, two integration operators will be used:

- Operator H: Receives two pieces as an argument and produces the minimum rectangle capable of covering such pieces, horizontally arranged, side by side. The dimensions of such possible minimum rectangle (L_h and W_h) are obtained as follows:
- L_h=large (p_1)+large_(p_2)
- W_h =max (width_(p_1),width_(p_2))
- Operator V: Receive two pieces as an argument and produces the minimum rectangle capable of covering such pieces, vertically arranged, side by side. The dimensions of such possible minimum rectangle (L_h and W_h) are obtained as follows:
- L_h=max(large (p_1),large_(p_2))
- W_h = width_(p_1) + width_(p_2)
- Graphically, both operators can be understood in the following manner:

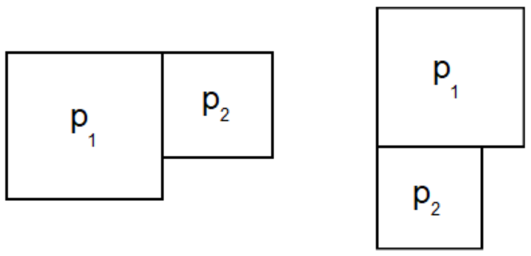


Figure 2: Pieces after applying the p1p2H and p1p2V operation respectively.

3.5 Interpretation of a Solution

This structure will serve both for definition of the solution known as "nest's egg" in the Cuckoo Search algorithm and for "individual chromosome" in the genetic algorithm. Using the operators of the previous section, the solution of a real case can be determined.

For example, if the solution is "1 5 H 7 6 V 2 H 4 H V 9 8 V 3 V H," start by taking the components from left to right in a stockpile until the emergence of a V and H operator, which will be responsible for integrating said components (i.e., pieces). See the figure below:

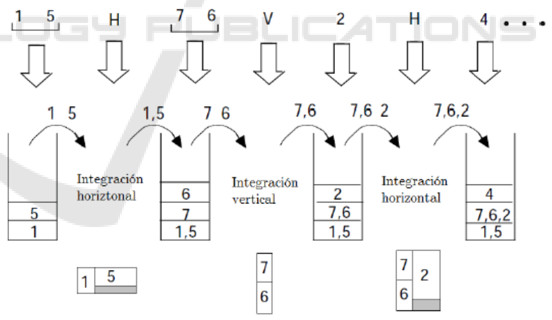


Figure 3: Pieces after applying the p1p2H and p1p2V operator simultaneously.

If we follow the steps through the end, we will obtain the following arrangement (in a binary tree format) where arrows indicate the order in which guillotine cuttings will be made:

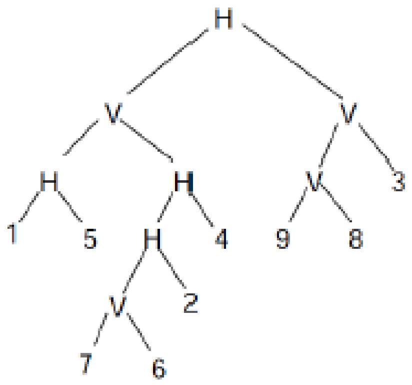


Figure 4: Representation as a binary tree.

3.6 Treatment of Defects in Pieces

Once the pieces are distributed in stocks, the main idea of this algorithm is to follow/walk through each of the stocks to verify that none has fallen in the regions with defects. If this is the case, the piece affected must be placed in one stock, so as to fulfil the order. Immediately the algorithm's pseudo code is shown for the stock's distribution:

```
Algorithm Defects(listStocks)
1. For each stock from listStocks do
     a. CalculateAbsolutePositions (stock.Tree, 0, 0)
     b. listPieces_Stock = ConvertToList(stock.Tree)
     c. For each piece from listPieces_Stock do
           i. listDefects = ObtainListDefects(stock)
           ii. If FoundInList(piece, listDefects) then
                 1. InsertPieceInOneStock(piece, listStocks)
               End if
        End for
   End for
2. End Algorithm Defects
```

Figure 5: Representation as a binary tree.

Each of the pieces of the stocks is followed to verify their positions. Absolute positions are calculated from pieces in a stock until the moment they were located in a relative manner (with respect to another block). The binary tree is followed/walked through again, in any order. The pieces in one stock are converted into a list. For each piece a list of defects related is obtained in a loop. It is verified that one piece is not intersected with one stock (both represented as pieces), comparing their absolute positions. If a piece falls into a region with defect, this is inserted in the stock immediately available.

3.7 Proposed Algorithm

The algorithm is based on the following main rules:

- Each cuckoo lays one single egg at a time and place it on the nest chosen randomly.
- The best nests with the highest egg quality will pass to the next generation.
- The number of nests available in each generation (n) is predefined, and eggs will be assessed (discovered) and removed if a pa probability is lower than the total nests. These nests (containing the eggs removed) will be replaced with new eggs each generation (iteration).

The proposed algorithm may be expressed as follows:

```
Algorithm CuckooSearch(numMaxGenerations, p_a)
  1. listNests = CalculateInitialNests(numNests);
  2. For i = 0 to numMaxGenerations do
       a. nestCuckoo_1 = SelectRandomNest(listNests)
       b. nestCuckoo_1 = UpdatePathLevyFlight(nestCuckoo_1)
       c. nestCuckoo_2 = SelectRandomNest)
       d. If (Fitness(nestCuckoo_1) > Fitness(nestCuckoo_2)) then
             i. nestCuckoo_2 = nestCuckoo_1;
          End if
       e. RemoveWorstAndGenerateNests(listNests, p_a)
       f. RankLowestToHighestFitness(listNests)
     End for
  3. Return listNest[0]
End Algorithm CuckooSearch
```

Figure 6: Algorithm Proposed.

Detail explanation:

- Line 1: A list of initial nests is calculated randomly.
- Line 2: An i variable is created to iterate each generation.
- Line 2-a: A random nest is selected from the list of nests.
- Line 2-b: A modification is done to the selected nest by means of a function following the Levy distribution.
- Line 2-c: A random nest is selected from the list of nests.
- Line 2-d: If the nest modified is better than the other nest, then keep the modified nest.
- Line 2-e: A fraction (p_a) of the worst nests is removed because they have been found.
- Line 2-f: The list is arranged by fitness from the lowest to the highest. Remember that this function of the case we are proposing is as follows:

$$\text{Fitness} = \left(1 - \frac{\sum_i^N \text{areaPieces}_{\text{stock}_i}}{\sum_i^N \text{area}_{\text{stock}_i}}\right) \times 100 \qquad (2)$$

- Line 3: Return the main nest, which is the one occupying the first position.

4 EXPERIMENTATION

As previously indicated, a comparison was made with a genetic algorithm also developed by the authors in an attempt to try the quality of results yielded by the Cuckoo algorithm.

For 40 test instances taken from a real Peruvian ceramic manufacturer, the following table shows the results of total waste in units of area (square meters/centimeters), a value resulting from the respective objective function of each of the algorithms:

Table 4: Example of List of Stocks.

Sample	Objective function Genetic Algorithm	Objective function Cuckoo Search Algorithm
1	46.07	55.06
2	36.67	50.72
3	40.00	43.29
4	28.15	31.06
5	43.98	46.37
6	29.51	30.16
7	34.63	38.08
8	36.24	47.24
9	40.49	51.71
10	38.22	48.19
11	45.73	46.85
12	39.18	52.63
13	39.67	48.23
14	37.78	51.55
15	49.31	57.23
16	41.62	52.62
17	41.51	41.64
18	35.18	40.45
19	46.55	51.77
20	37.02	38.05
21	39.84	47.27
22	38.05	40.84
23	41.25	53.88
24	45.50	53.97
25	39.46	43.26
26	40.60	49.41
27	35.02	44.60
28	48.41	52.29
29	45.18	51.86
30	36.13	47.49
31	47.18	54.70
32	36.87	41.59
33	45.32	50.05
34	40.10	40.21
35	45.22	51.33
36	45.80	48.61
37	49.99	53.09
38	36.22	46.70
39	43.04	53.31
40	46.39	59.30

5 CONCLUSIONS

Although it is possible to find accurate algorithms in the literature allowing for the solution of the 2D cutting problem, most of said algorithms are computationally infeasible with large instances of the problem. According to Kallrath, who made experiments in production lines of the paper sector for 25 types of pieces (molds), the algorithm is unfeasible (Kallrath et al, 2014).

Likewise, based on the results of the numerical experimentation, the mean of the objective function and the solution generated by the genetic algorithm have proven to be significantly lower than the mean of the objective function of solutions generated by the Cuckoo Search algorithm. It must therefore be concluded that for the set of data used the genetic algorithm has a better performance than the Cuckoo Search algorithm. This can be explained bearing in mind that the latter algorithm has random movement (Levy's flight) as the phase of improvement that in most cases do not improve the real solution, unlike the phase of improvement of the genetic algorithm in which crossing, and mutation operations are indeed enhanced.

Finally, the genetic and Cuckoo Search metaheuristic algorithm designed in this research to solve the cutting problem may be easily adjustable to other cutting problem variants (properly modifying the structure of the solution).

REFERENCES

Moore, R., Lopes, J., 1999. Paper templates. In *TEMPLATE'06, 1st International Conference on Template Production*. SCITEPRESS.

Smith, J., 1998. *The book*, The publishing company. London, 2nd edition.

Arbib, C., Marinelli, F., Pizzuti, A., and Rosetti, R., 2018. A Heuristic for a Rich and Real Two-dimensional Woodboard Cutting Problem. In *7th International*

Conference on Operations Research and Enterprise Systems (ICORES 2018), pages 31-37.

Booth, R., 2017. *Procesamiento y caracterización de materiales cerámicos refractarios del sistema ZrO_2-CaO-MgO-SiO_2*. Doctoral thesis. Universidad Nacional de La Plata, Buenos Aires.

Du, K., and Swamy, M., 2016. *Search and Optimization by Metaheuristics*. Springer, Basel.

Dyckhoff, H., 1990. A typology of cutting and packing problems. In *European Journal of Operational Research*, volume 44, pages 145–159.

Iglesias, A., Gálvez, A., Suárez, P., Shinya, M., Yoshida, N., Otero, C., Manchado, C., and Gomez-Jauregui, V., 2018. Cuckoo Search Algorithm with Lévy Flights for Global-Support Parametric Surface Approximation in Reverse Engineering. In *Symmetry,* volume 10, page 58.

Kallrath, J., Rebennack, S., Kallrath, J., and Kusche, R., 2014. Solving real-world cutting stock-problems in the paper industry: Mathematical approaches, experience and challenges. In *European Journal of Operational Research*, volume 238, pages 374–389.

León, C., Miranda, G., Rodríguez C., and Segura, C., 2007. 2D-cutting stock problem: a new parallel algorithm and bounds. In *European Conference on Parallel Processing (EURO-PAR 2007)*, pages 795-804.

Lodi, A., Martello, S., and Vigo, D., 1999. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. In *INFORMS Journal on Computing*, volume 11, pages. 345-357.

Pastor, J., 2002. *Fractura de materiales cerámicos estructurales avanzados*. Doctoral thesis. Universidad Complutense, Madrid.

Talbi, E., 2009. *Metaheuristics: from design to implementation*. John Wiley & Sons, New Jersey.

Tupia, M., Cueva, R., and Guanira, J., 2017. A bat algorithm for job scheduling in ceramics production lines. In *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS 2017)*, pages 266-270.

Tupia, M., Cueva, R., and Guanira, J., 2013. A GRASP algorithm with 2-opt improvement for job scheduling enviroment in ceramics production lines. In *AISS: Advances in Information Sciences and Service Sciences*, volume 5, number 15, pages 1-10.

Wäscher, G., Haußner, H., Schumann, H., 2007. An improved typology of cutting and packing problems. In *European Journal of Operational Research*, volume 183, pages 1109–1130.

Yang, X., and Deb, S., 2009. Cuckoo search via Lévy flights. In *IEEE Nature & Biologically Inspired Computing*, pages 210-214.