An Improved 2D Glass Cutting Solution with Genetic Algorithms

Mustafa Zahid Gürbüz
Yıldız Technical University
Istanbul, Turkey
mzahidgurbuz@gmail.com

İbrahim Emiroğlu
Yıldız Technical University
Istanbul, Turkey
emir@yildiz.edu.tr

Selim Akyokuş
Doğuş University
Istanbul, Turkey
sakyokus@dogus.edu.tr

Abstract—**2D glass cutting is an important problem for glass manufacturers. The objective of 2D glass cutting is to minimize the amount of waste when cutting a whole glass sheet into several pieces according to given cutting orders. In this paper, three different algorithms applied for the solution of 2D glass cutting problem. The performances of solutions are compared. The genetic algorithm with initial population strategy improves the performance of the algorithm and provides better results.**

Keywords- Genetic algorithms, cutting, optimization, flat glass.

I. INTRODUCTION

Glass production manufacturers deal with two different kinds of productions. One is the production of hollow glasses like bottle, cups, and other similar shaped glasses. Another kind is the production of flat glasses like window, mirror, and so on [1]. 2D flat glass cutting problem is a special case of general problem called 2D packing or stock cutting problem. In glass production, the objective is to minimize the amount of waste while cutting a glass sheet into requested pieces.

The algorithmic complexity of flat glass cutting problem is classified as non-deterministic polynomial (NP Hard). So the optimal solution of the problem couldn't be found in an acceptable amount of time. The execution time of the solution increases exponentially as the number of pieces increases. For many years, researchers have proposed various kinds of solutions to this problem. The solution methods can be divided into two major categories: deterministic and heuristic.

Deterministic methods consist of linear programming [2][3], integer programming with branch and bound [4], dynamic programming [5] and so on. In the linear programming approach, the objective is to minimize waste while cutting a whole glass sheet into pieces [2][3]. Integer programming is a variation of linear programming. It eliminates of inappropriate results from solution space. For example, there should not be a fractional result while cutting requested pieces. With deterministic methods, all possible cutting patterns have to be generated in order to find optimal solution. Therefore, the execution time of these algorithms grows exponentially with the problem size. These methods are usually complex and require extensive computation time, and are only practical for small problems.

Many of the computationally extensive NP-hard problems are solved by using heuristic methods. Heuristic methods are used when it is difficult to find an optimal solution. The heuristic methods produce near optimal solutions. In recent years, many heuristic based methods have been proposed. In 2D glass cutting problem, heuristic methods include Bottom left [6], bottom left fill [11][12], First fit [7][8], genetic algorithm [9][10]. In this paper, we apply a heuristic method to find a near optimal solution to the flat glass cutting problem.

Cutting problems can be divided into two categories according to the shapes of pieces. The shapes of pieces can be regular shaped (like rectangular or round) or irregular shaped (like asymmetric or non-convex). Rectangular shaped glass cutting can also be divided into two categories as guillotine and non guillotine cutting. In guillotine cutting, the cutting process continues until the end of the entire sheet of glass. It means that it cuts glass sheet on the parallel edges of rectangular pieces in one way (figure 1). If there is no cutting from one edge to other opposite edge, then it is called non-guillotine cutting (figure 2).

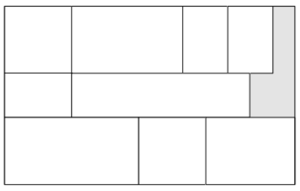


Figure 1. Guillotine rectangular stock cutting

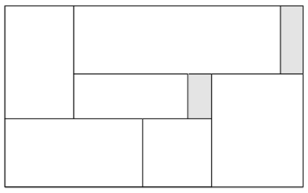


Figure 2. Non- guillotine rectangular stock cutting

There are two major problems in glass cutting. One problem is the scheduling of the orders. The system should select the required orders in an optimum way depending on their priorities. Once the orders are selected and scheduled, the second problem is to cut selected orders with minimum waste of glass. In this study, we deal with the second type of problem

978-1-61284-922-5/11/$26.00 ©2011 IEEE

and assume that guillotine cutting is used. We applied three heuristic methods in order to solve glass cutting problem. These heuristic methods are:

- First fit algorithm (FF)
- Bottom left algorithm (BL)
- Genetic algorithm with bottom left (GA-BL)

The algorithms input the following parameters.

- N is the number of different sized pieces
- k_i is the number of pieces in each different size (i=1,2,..N)
- w_i, h_i is the width and height of i^th piece (i=1,2,..N)
- W,H is width and height of the given glass sheet to be cut

Table I shows an example of cutting request order that is given as an input to the algorithms. In this order set, there are 22 pieces with N=4 different sizes.

TABLE I. AN EXAMPLE OF ORDERS

Piece no	Width w_i	Height h_i	Quantity k_i
1	60	14	12
2	71	112	5
3	153	201	2
4	50	105	3

II. FIRST FIT ALGORITHM

The first fit algorithm first sorts the requested pieces according to their height and then sorts by their width. The pieces are cut according to their sorting sequence. This algorithm is usually called as First fit decreasing height decreasing width (FFDHDW). The flow chart of FFDHDW algorithm is shown in Figure 3.

Korf used first fit algorithm in his study[8]. In 1974, Johnson mentions about the complexity of the first fit algorithm. The complexity of this algorithm is O(nlogn) [7].

III. BOTTOM LEFT ALGORITHM

Bottom left algorithm tries to cut pieces beginning with bottom left of a glass sheet. Each time the pieces is cut from bottom left if there is enough space on the given glass sheet [6].

The bottom left algorithm in some cases leaves gaps among the pieces. The normal algorithm determines the pieces to be cut in order without checking any gaps that is left before. Bottom left Fill algorithm includes a function that checks the gaps among the pieces and removes those gaps if possible [11][12]. Figure 4 shows a sample layout of pieces generated by bottom left fill algorithm. The bottom left fill algorithm is illustrated in Figure 5.

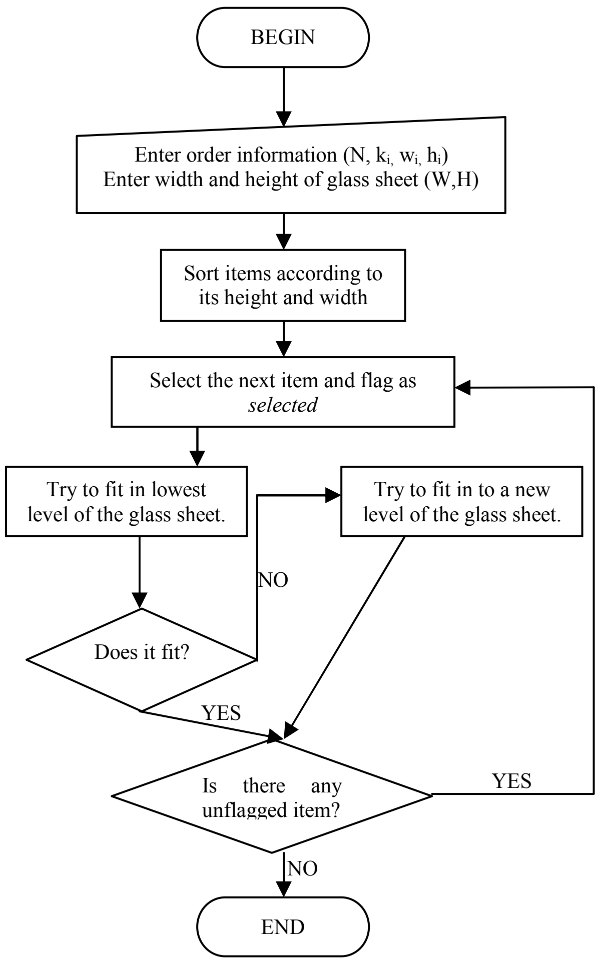

BEGIN
Enter order information (N, k_i, w_i, h_i)
Enter width and height of glass sheet (W,H)
Sort items according to its height and width
Select the next item and flag as *selected*
Try to fit in lowest level of the glass sheet.
Try to fit in to a new level of the glass sheet.
NO
Does it fit?
YES
Is there any unflagged item?
YES
NO
END

Figure 3. Flow chart of First fit decreasing height decreasing width (FFDHDW) algorithm

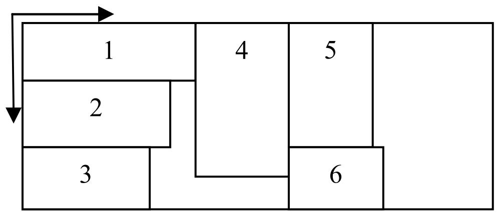


Figure 4. Bottom left fill algorithm sample

IV. GENETIC ALGORITHM

Genetic Algorithm (GA) is first described by Holland in 1975 [8]. Jacob [6] used Bottom left algorithm with genetic algorithm in 1996 on his study. Goldberg is used this algorithm on packing and cutting problems [9]. And many researchers made some improvement on genetic algorithm. Genetic algorithm is based on Darwin's evolution theory. Chromosomes pullulate by crossover. And good individuals pullulate by crossover of the good individuals. And bad individuals die by the natural selection.

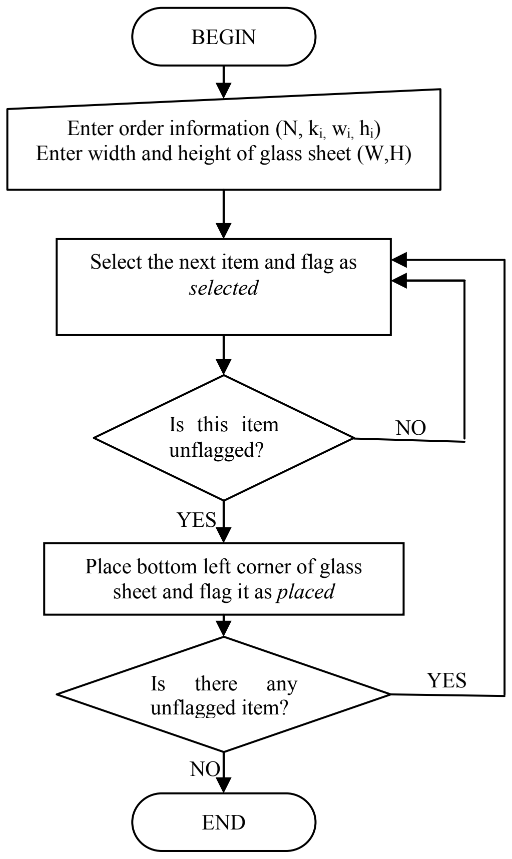

BEGIN
Enter order information (N, k_i, w_i, h_i)
Enter width and height of glass sheet (W,H)
Select the next item and flag as *selected*
Is this item unflagged?
NO
YES
Place bottom left corner of glass sheet and flag it as *placed*
Is there any unflagged item?
YES
NO
END

Figure 5. Flow chart of Bottom left Fill (BLF) algorithm

Genetic algorithms are usually used for complex optimization problems that do not require exact solution. It can be applied to solve many complex problems independent from problem area. In genetic algorithms, the most important thing is how to represent a problem using chromosome data structure and how to build coding scheme for genes. Binary or permutation encoding is generally used to encode genes of a chromosome.

In binary encoding, every chromosome is a string of 0's or 1's as in Figure 6. Each bits represents a binary variable in a problem domain.

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

Figure 6. Binary encoding of chromosomes

In permutation encoding, every chromosome is a string of numbers, which represents number in a sequence. Permutation encoding is usually used in ordering problems like cutting problems.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

Figure 7. Permutation encoding of chromosomes

Genetic algorithms try to find new and better solutions to a problem. The quality of each solution is measured by a fitness function.

A. Encoding of Chromosomes

The first step in genetic algorithm is to transform a real problem into chromosome data structure as in Figure 7. Each chromosome represents a different solution to a problem. Genetic algorithms begin with a set of solutions (chromosomes) called population. At the beginning, population is usually generated randomly.

In our study, we used permutation encoding and each element (gene) in chromosome represents the order of pieces to be cut. For example, if we have 9 pieces as in Figure 7, the order of genes gives the pieces to be cut.

B. Selecting Population

Population is a set of chromosomes that represents solutions. Solutions are taken from one population and used to generate a new population. The objective is to have a better population than the old one. The number of chromosomes in a population is called as population size. Population size is given as an input. In this study, we run the algorithm with the population sizes of 25, 50, 75 and 100.

In genetic algorithms, the next population is generated from previous population. Generally, initial population is generated randomly. In this study, in order to improve the solution of the problem, the solution of bottom left algorithm and the solution of the first fit algorithm is added as initial chromosomes to the genetic algorithm. This provides that at least some of our initial population has good solutions. The rest of the chromosomes in the population are generated randomly.

A new population, called as generation, is formed at each iteration of genetic algorithm. The maximum number of generations is given as an input to the algorithm. In this paper we assumed maximum 1000 generations to stop the algorithm.

C. Fitness function

Fitness function is very important in order to determine the optimality of the solution. Fitness function depends on problem type. To find a optimum solution, fitness function must be defined properly.

The value returned from fitness function is named fitness value. Fitness value must be calculated for each chromosome at each iteration. The optimality of Chromosomes depends on their fitness value. The chromosomes with the better fitness values are the better chromosomes (individuals) in the population. As explained in the next section, the better chromosomes are selected to generate a new solution set at each step.

In glass cutting problem, the objective is to reduce amount of wasted glass during production. Therefore fitness function is defined as a function of the wasted amount. We used the following fitness function:

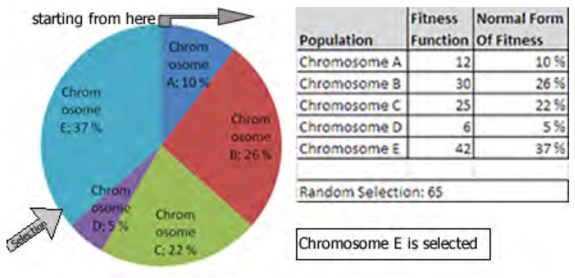
$$fitness = \frac{\sum_i w_i . h_i}{W.H}$$

Where W is the width of a whole glass sheet to be cut, H is the height of space occupied with the processed pieces, and w_i and h_i are the width and height of each of the processed pieces until that step.

D. Chromosome Selection

At each iteration of genetic algorithm, a part of population is selected to generate a new and better population using crossover and mutation operations. To make a crossover, pairs of chromosomes are needed. Chromosomes with better fitness values are usually selected to generate new chromosomes.

In this paper, we use roulette wheel method [14] in order to select the chromosomes to make crossover. In this method, the fitness value of each chromosome is normalized so that the sum of fitness values of all chromosomes is 100. The chromosomes are distributed around the wheel by their normalized fitness values. Then, a random number between 0-100 is generated and this random number is used to select a chromosome from the wheel as shown in Figure 8. The chromosomes with the higher fitness values have greater area on the wheel. Therefore, the probability of selection of the better chromosomes is higher than other chromosomes.



Population	Fitness Function	Normal Form Of Fitness
Chromosome A	12	10 %
Chromosome B	30	26 %
Chromosome C	25	22 %
Chromosome D	6	5 %
Chromosome E	42	37 %

Random Selection: 65

Chromosome E is selected

Figure 8. Roulette wheel example

E. Crossover

After a pair of chromosomes is selected, new individuals are generated by using crossover operation. Genes of the two chromosomes are interchanged during crossover operation. There are various kinds of crossover methods. The most popular ones are single point and uniform crossover methods.

Figure 9 shows single point crossover where a point is chosen to swap the genes of chromosome after starting from that point. The previous genes of the chromosome before the chosen point are unchanged.

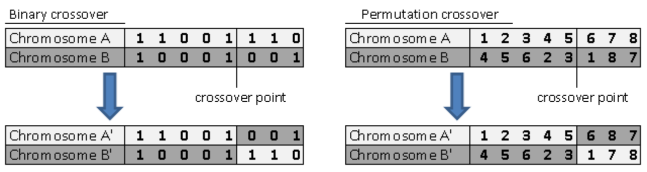

Binary crossover
Chromosome A 1 1 0 0 1 1 1 0
Chromosome B 1 0 0 0 1 0 0 1
crossover point
Chromosome A' 1 1 0 0 1 0 0 1
Chromosome B' 1 0 0 0 1 1 1 0
Permutation crossover
Chromosome A 1 2 3 4 5 6 7 8
Chromosome B 4 5 6 2 3 1 8 7
crossover point
Chromosome A' 1 2 3 4 5 6 8 7
Chromosome B' 4 5 6 2 3 1 7 8

Figure 9. Single point crossover

In Figure 10, uniform crossover method is illustrated. The genes are selected randomly from the pair of chromosomes. In Figure 10, the unshaded and shaded genes are selected randomly from chromosome A and B respectively to generate chromosomes A' and B'.

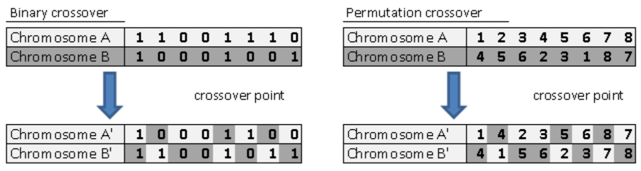

Binary crossover
Chromosome A 1 1 0 0 1 1 1 0
Chromosome B 1 0 0 0 1 0 0 1
crossover point
Chromosome A' 1 0 0 0 1 1 0 0
Chromosome B' 1 1 0 0 1 0 1 1
Permutation crossover
Chromosome A 1 2 3 4 5 6 7 8
Chromosome B 4 5 6 2 3 1 8 7
crossover point
Chromosome A' 1 4 2 3 5 6 8 7
Chromosome B' 4 1 5 6 2 3 7 8

Figure 10. Uniform crossover

In this paper, we used uniform crossover. A probability of crossover is given as a parameter to genetic algorithms that determines regeneration ratio. In each step, some of the chromosomes sometimes pass to the next population without crossover by this way. In this study, we used the value of 80% as crossover probability.

F. Mutation

After crossover operation, there is a chance that the values of some genes at particular positions will not change in new generations. To avoid this undesirable situation a mutation operator is used. The mutation operator enables the random alteration of some of genes. This action provides stepping over local optimum points. Figure 11 shows a sample mutation operation. In this study, only one gene of the chromosome is interchanged randomly with the mutation probability of 20%.

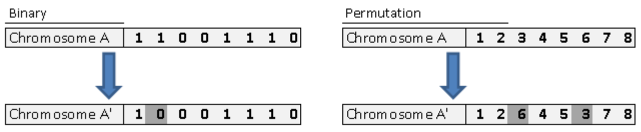

Binary
Chromosome A 1 1 0 0 1 1 1 0
Chromosome A' 1 0 0 0 1 1 1 0
Permutation
Chromosome A 1 2 3 4 5 6 7 8
Chromosome A' 1 2 6 4 5 3 7 8

Figure 11. Mutation operation

G. Elitism

Crossover and mutation operations may lead to the loss of good solutions. The elitist selection allows some of the better chromosomes from current generation to carry over to the next generation as unaltered. In our case, the elitist selection rate is selected as 10%. During each successive generation, 10% of the chromosomes with the best fitness values are retained without any changes.

Table II shows the summarization of the values of the parameters of the genetic algorithm that used in this study.

TABLE II. GA PARAMETERS.

Parameter Name	**Value**
Population Size	25, 50, 75, 100
Crossover Probability	80%
Mutation Probability	20%
Elitism	10% of the population size
Generation size	1000

V. EXPERIMENTAL RESULTS

In our study, we implemented bottom left fill (BLF) algorithm, first fit decreasing height decreasing width (FFDHDW) algorithm, and genetic algorithm (GA). To analyze these algorithms, 7 different glass cutting orders are generated randomly. These 7 orders include 10, 20, 40, 60, 100, 150, 200 pieces of various width and height. The whole glass sheet size is assumed to be 240 x 3000 units.

Figure 12 and Figure 13 shows layout of 20 pieces determined to be cut by BLF and FFDHDW algorithms respectively. In this sample run, allocation of pieces is different with the same amount of waste.

Figure 14 shows a solution produced by genetic algorithm for the same 20 pieces. In this case we obtained a waste amount of 16,06%.

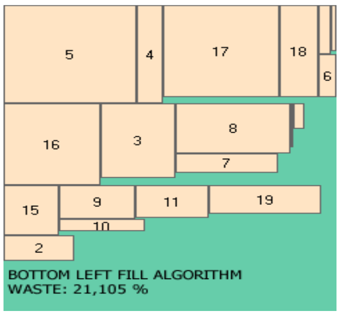


Figure 12. Bottom left algorithm layout for 20 pieces

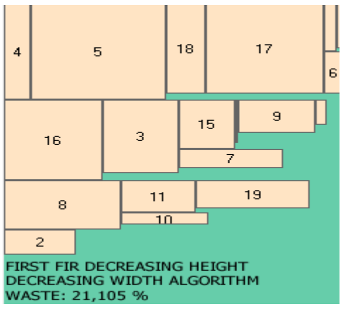


Figure 13. FFDHDW algorithm layout for 20 pieces

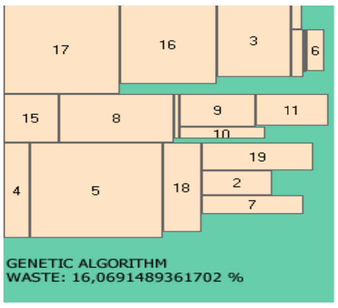


Figure 14. Genetic Algorithm layout for 20 pieces

Table III compares the minimum amount of waste calculated 10 times by applying BLF, FFDHDW, and GA algorithms on different cutting request orders with 10, 20, 40, 60, 100, 150, and 200 pieces. When applying genetic algorithms, different population sizes are selected in order to analyze and to decide the best population size. GA25, GA50, GA75, and GA100 stand for Genetic algorithm with the population size of 25, 50, 75, and 100 respectively.

The first step in genetic algorithms is to generate an initial population. This initial population is usually generated randomly. Initial population strategies can improve the performance of genetic algorithms [15]. In our genetic algorithm solution, an initial population strategy is applied to improve the performance of genetic algorithm. The results of two solution obtained by BLF and FFDHDW are added to the initial population. This provides that at least 2 good solutions are included in population before generating successive better solutions to the problem.

In Table III, GA25-RND stands for application of genetic algorithm on an initial population that is completely generated randomly without using initial population strategy. The column with title GA25 shows the result of waste obtained with using initial population strategy. As it is seen, the initial population strategy results in better values so that the amount of waste is reduced.

TABLE III. THE PERCENTAGE OF WASTE RESULTS FOR GA WITH THE POPULATION OF 25,50,75, AND 100 AND BLF, FFDHDW.

Pieces	GA25-RND	GA25	GA50	GA75	GA100	BLF	FFDHDW
10	31,9967	31,9967	31,9967	31,9967	31,9967	31,9967	31,9967
20	35,2160	18,1676	23,9493	25,9612	25,9612	26,7365	26,7365
40	18,9721	8,9098	11,0417	11,4186	10,0851	12,3471	12,3471
60	18,2023	7,3706	7,6517	8,8960	8,2089	9,7070	9,3033
100	16,0927	4,8359	5,4655	5,3761	5,9986	6,6130	6,6130
150	19,7437	4,2003	4,2584	4,4323	4,4323	4,8355	4,7781
200	13,3012	3,0775	3,5290	3,7531	3,2134	4,2426	4,1983

We also tested the genetic algorithm with initial population strategy on different population sizes: 25, 50, 75, and 100. As it can be seen in Table II, the best result is obtained with the population size of 25. So the increase of population size does not reduce the amount of waste. Application of GA algorithms on different population sizes always gives better performance over BLF and FFDHDW algorithms. When BLF and FFDHDW are compared, FFDHDW produces slightly better results.

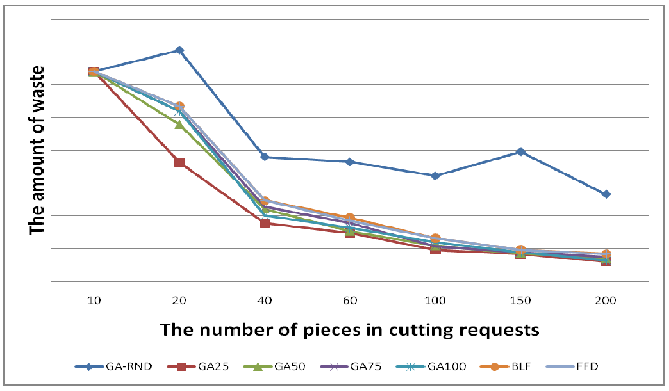


Figure 15. Graphics of results of BLF, FFDHDW

As shown in Figure 15, GA25-RND without initial population strategy produces the worst results among them. On the other hand, the performance of the other algorithms is better, and the best performance is obtained with GA25.

VI. CONCLUSION

2D glass cutting is a complex problem whose complexity is classified as non-deterministic polynomial (NP Hard). It is almost impossible to obtain an optimal solution to this problem because of the large number of alternatives in the solution space for large size problems. These kinds of problems are usually solved by using heuristic algorithms.

In this paper, we applied three different heuristic algorithms to solve the 2D flat glass cutting problem. We used an initial population strategy in genetic algorithm solution. This method improved the performance of the genetic algorithm and resulted in best results among the other applied algorithms.

This research work started after talking one of the glass manufacturers in Turkey. We learned that defects on the surface of glass sheets are an important issue when cutting the orders. The defected area should be excluded while cutting pieces. As a future work, we would like to add additional constraints like defects, apply other optimization methods, and develop new algorithms to the 2D cutting problem. Eventually, the glass manufacturer will use these developed algorithms if better solutions in both the amount of wasted glass and execution time are obtained.

REFERENCES

[1] F. Tooley, “Handbook of Glass Manufacture”, Glass Industry, 3rd edition, June 1985.

[2] P.C. Gilmore, R.E. Gomory, “A linear programming approach to the cutting stock problem (Part 1)”, Operations Research, Vol 9, pp. 849-859, 1961.

[3] P.C. Gilmore, R.E. Gomory, “A linear programming approach to the cutting stock problem (Part 2)”, Operation Research, Vol 11, pp. 863-888, 1963.

[4] A.H. Land, A.G. Doig, "An automatic method of solving discrete programming problems", Econometrica 28 (3), pp.497-520, July 1960.

[5] R. Sedgewick, “Algorithms in C++”, Addison-Wesley Professional; 2nd Edition edition, pp. 595-606, May 1992.

[6] S. Jakobs, “On genetic algorithms for the packing of polygons.”, European Journal of Operations Research 88, pp.165-181, 1996.

[7] D.S. Johnson, A. Demers, J. D. Ullman, M.R. Garey, R.L. Graham, “Worst-Case Performance Bounds For Simple One-Dimensional Packing Algorithms”, Siam J. Comput. 3,pp. 299-326, 1974.

[8] R.E. Korf, “A New Algorithm for Optimal Bin Packing”, AAAI-02 Proceedings, pp.731-736, 2002.

[9] J.H. Holland, “Adaptation in Natural and Artificial Systems”, University of Michigan Press, Ann Arbor, Mi, 1975.

[10] D.E. Goldberg, “Genetic Algorithms in Search, Optimization & Machine Learning”, Addison-Wesley Publishing Company, Inc, 1989.

[11] E. Burke, R. Hellier, G. Kendall,G. Whitwell, "A New Bottom-Left-Fill Heuristic Algorithm for the Two-Dimensional Irregular Packing Problem", Operations Research 54 (3),pp. 587–601, 2006.

[12] E. Hopper, B. C. H. Turton, “An Empirical Investigation of Meta-heuristic and Heuristic Algorithms for a 2D Packing Problem”, European Journal of Operational Research 128/1, pp. 34-57, 2000.

[13] E. Hopper, B. C. H. Turton, “A Genetic Algorithm For A 2D İndustrial Packing Problem”, Computers & Industrial Engineering 37, pp. 375-378, 1999.

[14] T. Back, “Evolutionary Algorithms in Theory and Practice”, Oxford Univ. Press, 1996.

[15] V. Toğan, A. T. Daloğlu, “An improved genetic algorithm with initial population strategy and self-adaptive member grouping”. Comput. Struct. 86, pp. 1204-1218, June 2008.