Research and development of algorithms for solving the two-dimensional irregular cutting stock problem

Maxim Poryvai*, mvporyvaj@lvk.cs.msu.ru,
Andrei Chupakhin*, andrewchup@lvk.cs.msu.ru,
**Lomonosov Moscow State University,*
Computational Mathematics and Cybernetics Department,
Moscow, Russia

***Abstract*—The cutting stock problem is relevant in many industries. It is NP-hard, and in practice finding its exact solution requires a lot of both time and computing resources. Therefore, when solving the cutting problem in general, approximate algorithms are often used, while exact algorithms are used only for special cases. This paper presents the results of an analytical review and a comparative analysis of various approaches to solving the cutting stock problem based on classic heuristic and brute force algorithms. Two algorithms for its solution have been developed and implemented: a greedy algorithm and a simulated annealing algorithm. An experimental study of the properties of the developed algorithms was carried out, as well as a comparison of the results of their work.**

***Index Terms*—two-dimensional irregular cutting stock problem, packing problem, nesting problem.**

I. Introduction

In many areas of the industry, for example, steel, leather, and textile, the problem often arises of cutting out given free-form blanks from large fragments of material. At the same time, it is required to group the locations of the contours of these blanks for cutting a fragment of material as tightly as possible to minimize waste (unused parts of this fragment). Thus, one of the studies [1] showed that saving only 1% of waste from each initial piece of material can prevent significant financial losses for any enterprise. Such problems are called cutting stock problems, or nesting problems in the literature.

II. Motivation

Nesting problems, as written above, are extremely important in the industry. But they belong to the class of NP-hard problems [2], so searching for their exact optimal solution is not very efficient in practice. But it also makes it meaningful to design, implement, try and compare new approximate algorithms for general cases and to search for more special cases to implement exact algorithms efficiently. This is what motivated this research and, as a consequence, the writing of this paper.

III. Related Work

Cutting stock problems are NP-hard, which means that it is reasonable to use heuristic algorithms to solve them. Existing algorithms were considered: natural (evolutionary, particle swarm, ant colony), tabu search, simulated annealing, and other heuristics. Exact algorithms can be used in special cases or small-sized tasks.

Cutting stock problems can vary significantly in terms of input data depending on specific requirements, for example, in terms of the parameters of the source material (the placement area can be finite, semi-infinite, and infinite) and the desired blanks (can be rectangles, convex / non-convex polygons, free-form shapes, with holes and without, with the possibility of rotation and reflection and without), restrictions on the remainder of the original fragment of the material (the remainder of the area (if it is finite) may not be taken into account, or additional conditions on the shape, size, etc. may be imposed on it).

However, various subproblems solved in these problems can be useful in solving any kinds of cutting stock problems. Let us consider in more detail the various found methods for solving cutting stock problems.

A. Natural Heuristic Algorithms

Quite a variety of solutions based on natural algorithms have been considered.

Shalaby and Kashkoush presented the particle swarm method in paper [3].

Thus, in their formulation of the problem, there is an area of fixed width and infinite length in one direction (a half-strip). Also, a set of arbitrarily shaped figures, but without holes, is given. The shapes can be rotated 90 degrees in any direction and 180 degrees. It is required to place all the shapes in the given area so that the length of the resulting rectangle is minimal.

In the beginning, a sequence of parts is randomly generated for placement (in the appropriate order). Then an attempt is made to place the parts by selecting them in a given order. The entire placement area is divided into squares of fixed size. Then all the parts are also approximated with similar parts consisting of the same squares. Now each part is tried to place so that it does not overlap with the already placed parts.

Then the best result is memorized and the process moves on to the optimizing particle swarm method. In this method, each "particle" is a location vector, and it also has a velocity vector. At each step, appropriate formulas are used to update values of these vectors for all particles from "swarm". In this way, the whole swarm moves as if to the optimal solution.

Applied to the problem, the arrangement vectors are the orders of placement of the pieces in the previous step, that is,

they consist of integers from one to the number of pieces. The velocity vectors are initially zeros. In the next steps, additional restrictions are also imposed on velocity vectors (module not more than half of the number of figures). The final vector of particle positions will be real numbers of different signs. The order of selection of figures on the placement corresponding to a given particle is equal to the order of the indices of the resulting position vector traversal in ascending order. For example, for a placement vector $(-1.0, 1.2, 0.5)$ the sequence will be $(1, 3, 2)$, where the index is the number of the figure assigned initially. The maximum iterations of the particle swarm method are set initially.

Singh and Ferres proposed in their paper [4] two evolutionary algorithms with completely different approaches to solve the two-dimensional irregular nesting problem and compare their results, showing that they are also completely different. The authors have also developed a new approach to estimating the quality of nesting maps, a new quality function for filling the nesting map to be minimized (hereafter, the fitness function).

In the problem, a rectangular area is given, divided into squares of equal size. All shapes also consist of such squares. There are 4 kinds of figures in all. They can be rotated by angles multiple of 90 degrees, as well as reflected. It is necessary to place the figures so that the value of the fitness function is minimal.

The fitness function is designed so that its value equals the sum of the empty squares, multiplied by 1000, plus the sum of the coordinate differences between all possible pairs of empty squares (the abscissa and ordinate differences are summed separately). Such a fitness function allows the algorithm to take into account not only the number of empty squares but also the compactness of their locations.

The idea of the first evolutionary algorithm is that initially the placement area is empty, and the pieces are tried to be placed on it one by one (the initial "generation" is formed randomly). In the crossover the offspring is obtained by alternating parts from the parents, the first parent is chosen randomly. A mutation is simply adding a random figure to a random place on the placement area. If an error occurs 5 times in a row, no mutation occurs.

The idea of the second evolutionary algorithm is that initially the placement area is full, and there are many overlapping shapes (the initial "generation" is formed randomly). Uniform crossover [5] is used in this approach. A mutation is simply changing the orientation or type of one of the randomly selected shapes.

Also, in each of the algorithms, the generations were constantly checked for irreducibility to a suboptimal solution, i.e. it was checked that the descendants were always as different as possible by the value of the fitness function, too close ones were eliminated leaving one of the doubles.

In Verhoturov's paper [6] it's proposed to solve the cutting problem using the ant colony algorithm, as well as the algorithm of simulated annealing.

The problem is considered in the following formulation: there is a packing area W and a set of geometrical objects P. It is assumed that the area of packing area is much larger than the area of all geometrical objects in the sum. Let us conventionally divide the region into an occupied part Q, where the figures are placed, and a remainder U. The essence of the problem is to minimize the uncovered area of the occupied part of the packing area, i.e. to place the given figures as densely as possible on the infinite area of two-dimensional space.

The author characterizes the quality of a nesting map (i.e. the image of a packing area with geometrical objects located on it) using the nesting coefficient defined as the ratio of the total area of placed figures to the area of the occupied part Q of the packing area.

The main idea of the ant colony algorithm is to implement the principle of collective intelligence. To find the extremum of the target function, the algorithm uses several agents (artificial ants) in parallel, which accumulate statistical information during the search. This information is accumulated in a publicly available databank and used by the agents independently of each other. Each agent acts according to the rules of the probabilistic algorithm and when choosing a direction it is guided not only by the increment of the target function but also by the statistical information reflecting the prehistory of the collective search. For the problem solved by the author, at each step, each agent constructs a set of admissible solutions, and then these solutions are compared for the presence of repeated components — they will be of higher priority in the next iterations.

The idea of the simulated annealing method is that a control parameter — temperature — is introduced that affects the magnitude of displacements between successive solutions to the optimization problem. A high temperature, at which the solution process begins, corresponds to large displacements, and a low temperature corresponds to small displacements. Large displacements are defined as such changes in solutions that make large changes in the values of the target function, and small displacements are small changes in the values of the target function. Large moves at high temperatures mean that the algorithm avoids a local minimum. The temperature slowly decreases to the minimum value during the solution process. Three types of displacements are used to change the location of objects: moving an object to a new location, moving two objects together, and changing the orientation of the object.

B. Other heuristic algorithms

Lopez-Camacho, Ochoa, Terashima-Marin, and Burke propose a different heuristic method for packing irregularly shaped figures into rectangular containers (sheets) [7].

The set of inputs is the set of figures; the size of the side of the square area (containers are square for simplicity); s_0 is the part of the container area, before filling which only one piece per container is put, and at filling which it is tried to either fill the container to the end with one piece or put 2 or 3 at once; boundary w is the step with which the allowed size

of the unused container space increases. The set of containers is infinite. The goal is to minimize the number of containers needed to place all the pieces (the pieces can be rotated by angles divisible by ninety degrees from their original position).

The packing method consists of a combination of selection and placement methods. The selection method is divided into several parts. First, the parameters s_0 and w are chosen. The algorithms are then applied sequentially: before filling the container at a fraction of s_0 one part is put each, and when filled at s_0 algorithm tries to either fill the container to the end with one part or put 2 or 3 parts at once to fill the container to the end. If unsuccessful, the algorithm tries to fill the container in the same way without using w of its volume, then $2w$, and so on. When filling a container, special algorithms are used to determine intersection and overlapping parts. The intersection of parts is defined as the intersection of any of their boundaries so that if one part is completely inside the other, the algorithm will not work. Therefore, a separate algorithm is then used to determine overlap.

Important improvements in the proposed packing method were: checking for special cases in almost all algorithms (for example, if rectangles bounding the parts do not intersect, then obviously the parts themselves do not intersect either), which gives an increase in the average speed of the program, since these special cases occur quite often for all cases; no retries to place the part on the sheet after at least one failed attempt (for CAD method this would be extremely time consuming, but it does not give

Verhoturov in his article [6] also considers the method of tabu search as a way to solve nesting problems.

The idea of the algorithm is not to stop at the local optimum but to continue the search, guided by the same rules, banning visits to already passed points from the neighborhood (reexamination of similar options). However, if the older information will not be removed from the list, then the performance of the algorithm will fall with the increasing number of iterations. Therefore, the length of the list of bans is limited from above by some constant.

Each time a figure is moved to another place, it is first looked at whether this figure is not in the list, and if it is, the figure is rearranged. After shuffling, the piece is put on the list. If the length of the list is already equal to the maximal possible length by this moment, the first element of the list (the one added before the others) is removed from it. The essence of the search method with bans is that it most likely makes no sense to rearrange a figure that has just (or just recently) already changed its position because in practice this often leads to a huge number of useless calculations.

C. Exact algorithms

Fischetti and Luzzi [8] in their paper propose to solve the cutting stock problem by bringing it first to the mixed-integer linear programming form. Mixed-integer linear programming differs from integer linear programming in that the integer condition must be satisfied for at least one input parameter, not for all.

The problem is considered by the authors in the following formulation. Given a set of polygonal figures, not necessarily convex, each of which has a certain reference point and a half-band with fixed width and potentially infinite length. The problem is to place all the pieces on the half-band, minimizing the length occupied by the pieces. This is identical to maximizing an efficiency function equal to the ratio of the total area of the pieces to the area of the final strip.

In this solution the authors use NFP (no-fit polygon) — a polygon that encloses any fixed figure in the set; sliding down it with an anchor point of an unfixed figure will not lead to the intersection of these figures (we get a touch), but moving into the interior by an infinitesimal value will lead to the intersection. IFT (inner-fit region) is also used — a polygon inside a fixed figure which, when slid by the anchor point of an unfixed figure, produces an overlay of the given figures, but which cannot be overlapped if we move outside by an infinitesimal value.

The authors divide the external area of the figure into pieces for convenience: each piece has one of the boundaries of one of the sides of the figure and goes to infinity.

The authors set the problem of cutting in the formulation of mixed-integer linear programming, setting the goal of minimizing the function of the length of the strip and some conditions (inequalities). Among them are the condition of non-intersection of figures, the condition that for each placed figure it is only one piece considered of the outer area of the fixed figure, and others.

The problem is further solved using the branch and bound method.

D. Review conclusions

Various approaches to solving cutting stock problems were considered. Exact approaches, such as integer and mixed-integer linear programming methods in combination with the branch and bound method, are not scalable to a large number of placed shapes: the upper bound does not exceed several dozen figures. The ant colony algorithm and the particle swarm algorithm have a large number of parameters, which negatively affects the computation time of these methods. Evolutionary algorithms need a lot of memory to store populations to work. Algorithms not based on well-known analogs are more difficult to study. Among the remaining algorithms, the simulated annealing algorithm that has only two parameters and does not require a lot of memory during its work was chosen first to be implemented. It was also decided to develop a greedy algorithm for further comparison of the results of the work. The NFP method was chosen for non-intersecting placement.

Some references to the literature are not as new, but this is because the essence of the algorithms applied to the cutting stock problem is always about the same, but in new works the essence of the algorithms often recedes into the background, giving way to additional minor refinements. In the review process, however, it was important to get ideas for applying known algorithms to nesting problems; the more subtle points

related to implementation will be considered when the basic algorithms are implemented.

IV. Problem Statement

A. Informal Statement of the Problem

There are an infinite number of rectangular sheets, the length and width of each sheet are given and the same. There is also a finite set of polygon shapes to be placed on the sheets. The figures do not have holes and can be rotated through angles that are multiples of 90 degrees from their initial position.

It is required to find such a layout of figures that: a) minimizes the number of sheets required to accommodate all the figures, b) the placement is as dense as possible.

B. Mathematical Statement of the Problem

Let there be $Q = \{q_i, i = 1, \ldots, M, \ldots\}$ - an infinite set of rectangular sheets, the dimensions of each sheet are a, b, $a \geq b$. The number of sheets used for figure placement will be denoted as M. There is also a finite set of polygon shapes $S = \{s_i, i = 1, \ldots, n\}$ to be placed on the sheets. Each polygon s_i is defined by a sequence of its vertices $s_{ij}, j = 1, \ldots, k_i$. The figures do not have holes and can be rotated by $\alpha = 90t$ ($t \in \mathbb{Z}$) degrees relative to their initial position.

If the figure s_j is placed on the sheet q_i, it will be written as $s_j \in q_i$.

Let us denote the convex hull for all placed figures on the sheet q_i as d_i, and the envelope for the placed figures as e_i. The area function of the figure x will be denoted as $sq(x)$.

It is required to find such a layout of figures on sheets that minimizes a fitness function:

$$FF = 1 - (A \cdot FF_a + B \cdot FF_b)$$

$$FF_a = \frac{\sum_{i=1}^{M-1} (1 - \sum_{s_j \in q_i} sq(s_j))/sq(q_i) + (1 - \sum_{s_j \in t_M} sq(s_j))/sq(e_M)}{M}$$

$$FF_b = \frac{\sum_{i=1}^{M} (1 - \sum_{s_j \in q_i} sq(s_j))/sq(d_i)}{M}$$

FF_a function is equal to the average over all sheets of the ratio of the area of all parts on the sheet to the area of the sheet, so is essentially responsible for minimizing the number of sheets. FF_b function equals the average over all sheets of the ratio of the area of all parts on the sheet to the area of the convex envelope bounding them, thus responsible for the density of placement. Thus, the task of multi-criteria optimization is reduced to the task of single-criteria optimization.

At the moment, it has been decided to use the values of coefficients A and B equal to 0.5, which leads to the normalization of the values of the fitness function to the segment $[0, 1]$. Therefore, the fitness function shows the quality of the resulting placement of figures on the sheets. Generally good coefficients are further planned to be chosen empirically.

C. Purpose of Research

The purpose of this study is to select the best algorithm according to the criteria a)-b) specified in the meaningful statement of the problem in each class of algorithms selected as a result of the analytical review, experimental study of algorithms, and comparison of the selected algorithms. The choice and subsequent comparison of the algorithms will be carried out according to the following main criteria: the running time of the algorithms on input data of different sizes (to determine the scalability of the algorithms) and the quality of the resulting solution, which is determined by a fitness function.

The result of this study will be a software library containing implementations of algorithms from different classes, solving the cutting stock problem in the above-mentioned statement, implemented in the Python programming language (some time-critical parts will be written using C++) and designed in a common style, with a unified hierarchy of classes. It is also possible to partially use code from open-source projects and modify it.

V. Implementation

A. General Design

At the moment the library [9] is implemented in the Python programming language because it allows to make the development faster and clearer, although to the detriment of the speed of the program. Nevertheless, some of the dependencies from additionally used libraries can be transferred to C++, as well as the rest of the code. Therefore it is planned to first implement the library in Python, and then partly translate it into C++.

Points (vertices) in the program are represented by tuples, polygon shapes are represented by tuples of their vertices. The set of shapes to be placed is initially specified by the shape: number dictionary. Each of the figures is initially specified by specific coordinates in the plane, but the actual placement of each figure will undergo transformations (consisting of rotations by a multiple of 90 and mirroring) and take into account offsets (translations) relative to the origin of coordinates for the initial figure coordinates. Therefore in the program, each figure will be represented as a tuple of three elements: figures in initial coordinates, coordinates of transformed figures in initial coordinates and displacement. Each of the shape placement sheets is a list of such three-element tuples corresponding to the shapes.

The class diagram is shown in $Fig.1$.

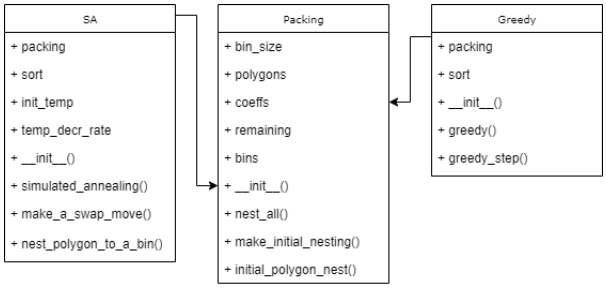

Fig. 1. Class diagram.

The main class of this library is the *Packing* class, whose fields are: *bin_size* - tuple containing bins width end height,

bins - sheets from the previous paragraph, *polygons* - polygons from the previous paragraph, without transformations and translation, *remaining* - dictionary of remaining figures and their counts, *coeffs* - fitness function coefficients.

Each of the algorithms has its own class: *SA*, *Greedy*.

The *Greedy* class has the following fields: *packing* - an element of the *Packing* class, whose nesting map is built by the algorithm, *sort* - the initial sorting type used: random or in descending area order. The methods of the *Greedy* class: *greedy()* - sorts the list of shapes and feeds the shapes one by one into the *greedy_step()* method, where the current shape is placed.

The *SA* class has the following fields: *packing* - an element of the *Packing* class, the nesting map of which is built by the algorithm, *sort* - the initial sorting type used. The methods of *SA* class: *simulated_annealing()* - sorts the list of shapes, generates the initial placement and contains the main loop of the algorithm. The *make_a_swap_move()* method is called inside this method, which randomly selects sheets and shapes for moving and calls *nest_polygon_to_a_bin()* method which makes the movement.

It is possible to visualize the resulting nesting map using *plot_packing()* and *plot_bin()* functions, based on the *MatPlotLib* module. An example of the visualization is shown in *Fig*.2.

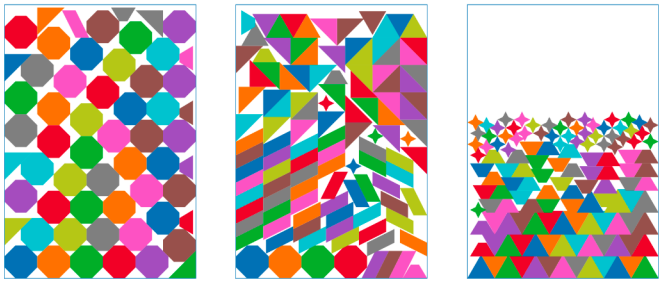

Fig. 2. Example of a result visualization.

The input data to the program are fed with a text file of a special format (description in README.md on GitHub), and the output file contains the text output of *packing.bins*.

In both algorithms placement strategy is bottom-left.

B. Greedy Algorithm

One of the simple but quite effective algorithms turned out to be the greedy algorithm. In this library, it is implemented as follows. First, all figures are sorted either in descending order of area or in random order. Then the figures are placed on the sheets in this order. Then a list of all possible transformations is made for the current figure. Then the most appropriate placement (according to the bottom-left strategy) is calculated for each of the transformations of this figure on each of the available sheets, using NFP [2], and the variant that gives the largest fitness gain is chosen. If none of the transformations of a given figure can be placed on any of the current sheets, it is placed in the lower-left corner of the newly created sheet.

An example of the algorithm working process is shown in *Fig*.3.

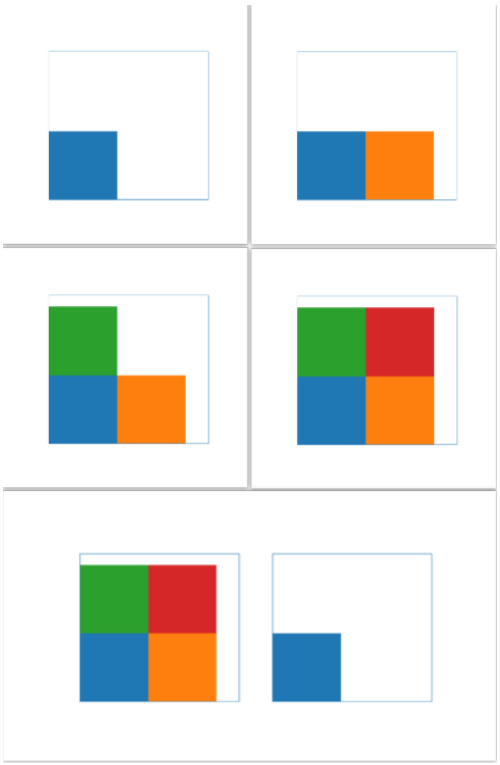

Fig. 3. Greedy method working example.

C. Simulated Annealing Algorithm

Another method currently implemented is the simulated annealing algorithm. The algorithm is based on the simulation of the physical process that occurs during the crystallization of a substance, including the annealing of metals. It is assumed that the atoms have already lined up in a crystal lattice, but transitions of individual atoms from one cell to another are still permissible. It is assumed that the process proceeds at a gradually decreasing temperature. The energy function of the system is also defined, and a stable lattice corresponds to a minimum of this function. The transition of an atom from one cell to another occurs exactly if it lowers the energy of the system, and with some probability, if it does not, and this probability is greater when the energy difference is closer to zero and it also decreases with temperature decreasing.

In the current implementation of this method, an initial placement is first generated, which differs from the greedy algorithm in that the shapes are not transformed and are placed only in the last of the current sheets each time. Next, the initial temperature T and the step of temperature decreasing are set as parameters. Energy of the system $E = 1000000 \cdot (1 - FF)$, so it is normalized by $[0, 1]$. Then, until the temperature reaches zero, a while loop is started, at each iteration of which 2 random figures from the current placement are selected and swapped their placement sheets with each other (the specific placement is also selected by the bottom-left strategy with

the NFP), also with the random transformation. If the shapes cannot be swapped correctly, the temperature is reduced by a tenth of a step. Otherwise, the difference of the energies ΔE before and after the exchange is calculated. If it is negative, the exchange is successful, and the temperature decreases by the step value. If it is non-negative, the exchange occurs with a certain probability: in case a randomly generated decimal from the range $[0, 1]$ is less or equal than $\exp(-\Delta E/T)$.

VI. Testing Results

A special generator function was written to make the test set. The following parameters for the generator function were used for the experiments:

- sheet sizes: 70x100;
- different figures: rectangle 9x4, equilateral triangle of side 10, isosceles trapezoid with bases 9 and 5 and height 5;
- initial number of each of the figures: 10;
- final number of each of the figures: 505;
- incremental step of number of each figure: 5;

This set of parameters was applied to both algorithms and to both sorting methods. Initial temperature equal to 500 and temperature decreasing step equal to 1 were used in the SA algorithm. The running times of the algorithms on these sets and the obtained values of the fitness functions are shown in $Fig.4$ and $Fig.5$, respectively. The stability study of the simulated annealing algorithm and the profiling of the algorithms will be carried out after their refinement.

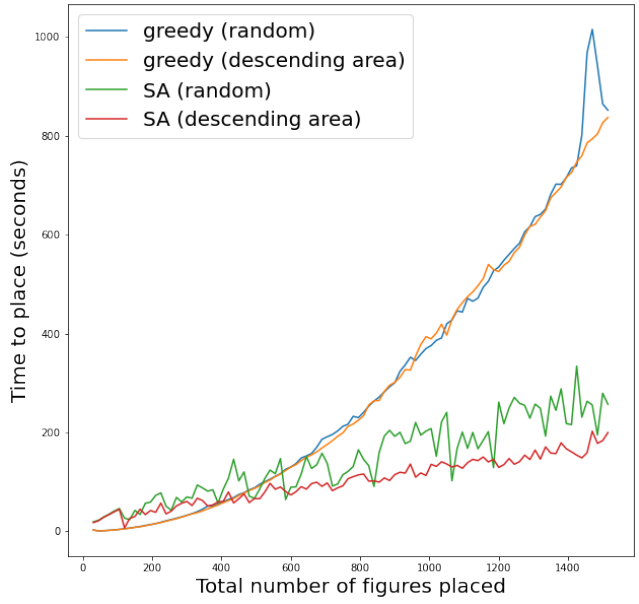

Fig. 4. Working times of the algorithms.

VII. Conclusion

According to the results of testing the algorithms, it can be concluded that the greedy algorithm at the moment gives a better result, while the time of its work increases approximately exponentially with the number of pieces placed. The strategy

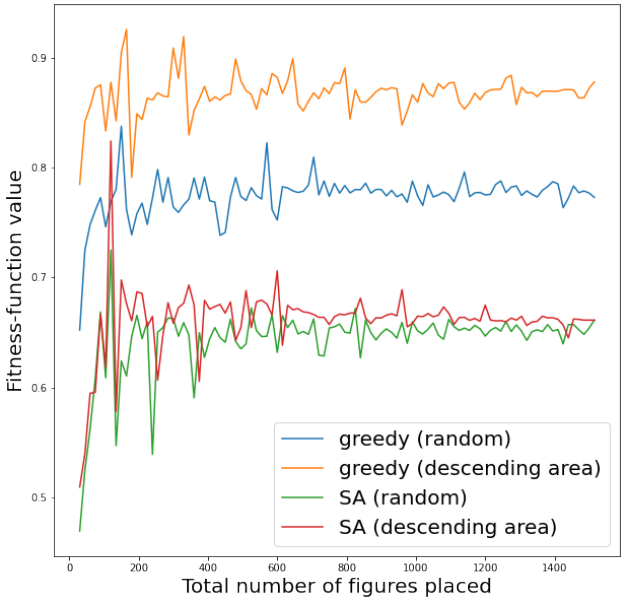

Fig. 5. fitness function values of the algorithms.

of sorting shapes in descending area order shows itself better than the random one.

The simulated annealing algorithm does not yet give the expected result, as can be seen in the graph. Perhaps the method of transition to the new cutting map should be modified, as well as the energy function.

Next, it is planned to refine the simulated annealing algorithm and to implement more algorithms (e.g. a genetic algorithm, a tabu search algorithm). Then some time-critical parts of the library will be translated into C++ programming language.

References

[1] CEPI Key Statistics Report 2012, Confederation of European Paper Industries, URL: https://web.archive.org/http://www.cepi.org/node/16197

[2] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, San Francisco, 1979, x + 338 pp.

[3] M. Shalaby , M. Kashkoush, "A Particle Swarm Optimization Algorithm for a 2-D Irregular Strip Packing Problem," American Journal of Operations Research. 3. 268-278. 10.4236/ajor.2013.32024 (2013).

[4] G. Singh and H. Lavista Ferres, "2D Packing problem with irregular shapes," (2014). URL: https://usnd.to/LF3O

[5] E. Eiben, J. E. Smith, "Introduction to Evolutionary Computing," A, Springer, 2003

[6] M. Verhoturov, "Zadacha nereguljarnogo raskroja figurnyh zagotovok: optimizacija razmeshhenija i puti rezhushhego instrumenta,", Vestnik Ufimskogo gosudarstvennogo aviacionnogo tehnicheskogo universiteta, vol. 9, no. 2, 2007, pp. 106-118

[7] E. Lopez, G. Ochoa, H. Terashima-Marin, E. Burke, "An effective heuristic for the two-dimensional irregular bin packing problem," Annals of Operations Research. 206. 241-264. 10.1007/s10479-013-1341-4 (2013).

[8] M. Fischetti, I. Luzzi, "Mixed-Integer Programming Models for the Nesting Problem," Journal of Heuristic, Vol. 15, No. 3, 2009, pp. 201-226.

[9] 2D-irregular-stock-cutting-problem-multi-algorithm-solution. URL: https://github.com/maxporyvay/2D-irregular-stock-cutting-problem-multi-algorithm-solution