UDC 004.02

SOLVING THE PROBLEM OF ONE-DIMENSIONAL CUTTING ON THE BASIS OF GENETIC ALGORITHM

Valentin Davydov, Olga Tarakhtiy
Odesa Polytechnic National University

Abstract. *The process of synthesis of algorithm for solving the problem of one-dimensional cutting of furniture profile is considered. The following are shown: the process of designing the necessary types of data, selection of criteria and synthesis of the target function, synthesis of the algorithm for solution search on the basis of genetic algorithm elements.*

Keywords: *One-dimensional cutting problem, genetic algorithm.*

Introduction

Any modern furniture production sooner or later faces the need to optimize the cutting process. Today, the market offers many cutting automation programs, both paid and free of charge. But not always the existing solutions meet the requirements of the manufacturer. Some programs do not cope well with large input data sizes. Some programs take too long to find a solution. Some offer solutions that are far from optimal. But one of the main disadvantages can be considered that these are fully completed stand-alone products. They are not designed to be integrated into an existing process. At the same time, the raw data for optimization can be the product of different design systems. The result of the optimization can be used to print cutting cards. In other words, the cutting tool should be seamlessly integrated into the existing production scheme and easily adaptable to established information flows.

1. Analysis of the state of the cutting task

In general, the cutting problem belongs to the class of Bin Packing Problem (BPP) [1,2]. Although it is known that the optimal solution exists and can be found by a complete enumeration of all variants, the computational complexity of the problem increases very rapidly with increasing dimensionality. Additional conditions and constraints also significantly increase the computational complexity of the problem.

In classical terms, the BPP problem can be formulated as follows. There are some containers of a certain size, and there is a set of parts to be packed into these containers. The goal is to pack all parts using the smallest number of containers. Let us consider this problem in the context of furniture production.

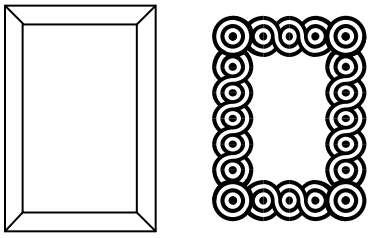

Fig. 1. Utilization of profiles in furniture component manufacturing.

The design of some furniture elements (doors, panels, frames, etc.) provides for finishing the perimeter of the part with a profile (Fig. 1). The profile can have a load-bearing function and/or a decorative function. As a rule, for one element it is necessary to cut two parts of size *x* and two of size *y*. Within one set the number of parts can be measured in tens and sometimes, depending on design decisions, can reach up to a hundred. In mass production, it may be necessary to optimize the nesting not for a single set, but for the entire batch.

The profiles used, from which parts are to be cut, may differ in shape, material and length. In the process of cutting parts, profile trimmings are left behind. There is a certain critical length L_{krit} . Cuttings longer than this length are suitable for further use. Cuttings shorter than this length go to waste.

Thus, an additional constraint is imposed on the classical formulation of the BPP problem: it is necessary to fill all containers so as to minimize unused pruning. That is, it is not enough to simply obtain a solution. It is necessary to choose the best solution from the whole set of solutions.

Today there are many approaches, methods and algorithms for solving BPP [3]. One of the effective approaches to solving the formulated problem is genetic algorithm. The purpose of this paper is to demonstrate the solution of the problem using

genetic algorithm (GA) on the example of one-dimensional nesting.

As noted above, each established furniture (and not only) production is characterized by its own scheme of information flows. The designer creates a 3D model. It is transformed into specifications, sets of cutting cards. The cards are printed and transferred to CNC operators for additional control. Electronically, the cards are converted into a specific set of CNC commands. When transferring information between individual process stages, files of various formats from unique to common formats can be used. The information flow diagrams may differ at each particular production facility, but they are always present. Therefore, first of all, it is necessary to analyze the existing information flows and determine the type and method of organization of input and output information.

2. Synthesis of the solution algorithm

In this task, we assume that the input and output information are represented by files in xml format.

The structure of the input file is as follows. All dimensions are in millimeters. The file contains elements: "**data**" - common root; "**list_profiles**" - descriptions of profiles available for cutting; "**list_details"** - description of parts to be made from profiles.

The list of profiles contains attributes: "cut_width" - kerf width; "**rest_min_length**" - minimum length of the **rest,** which can then be used in cutting.

In addition, the list of profiles contains one or more nodes "**profile**" containing the following data: "type" - type of profile, 0 - initial (whole) profile, 1 - profile residue; "**number**" - number of profile residue or 0 - for whole profile. The number is necessary to allocate the required residue for cutting; "quantity" - quantity of the profile of the given type and number; "length" - length of the profile.

The parts list contains an attribute "**order_name**" with the nesting number and one or more "**detail**" nodes with the following data: "**number**" - part number; "name" - part name; "length" - part length; "quantity" - number of parts; "comment" - additional data about the part.

Example of a source file.

```
<?xml version="1.0" encoding="windows-1251"?
<data>
  <list_profiles cut_width="4" rest_min_length="300">
    <profile type="0" rest_number="0" quantity="10" length="5500"/>
    <profile type="0" rest_number="0" quantity="20" length="5100"/>
    <profile type="1" rest_number="1001" quantity="1" length="1200"/>
    <profile type="1" rest_number="1002" quantity="2" length="800"/>
    <profile type="1" rest_number="1003" quantity="1" length="1040"/>
    <profile type="1" rest_number="1004" quantity="3" length="654"/>
    <profile type="1" rest_number="1005" quantity="4" length="754"/>
  </list_profiles>.
  <list_details order_name="1003">
    <detail number="003494130001" name="facade" length="697" quantity="8" comment=""/>
    <detail number="003494130002" name="facade" length="447" quantity="8" comment=""/>
    <detail number="003494140001" name="facade" length="1197" quantity="4" comment=""/>
    <detail number="003494140002" name="facade" length="447" quantity="4" comment=""/>
    <detail number="003494150001" name="facade" length="237" quantity="12" comment=""/>
    <detail number="003494150002" name="facade" length="597" quantity="12" comment=""/>
  </list_details>.
</data>.
```

Output file structure. The file contains elements: "**data**" - common root; "**list_plants**" - list of nesting cards.

The list of nesting cards contains the nesting number (previously passed the "**order_name**" attribute in the job file) - the "**order_name**" attribute.

In addition, one or more elements "**plan**". Element "**plan"** has attributes: "**number"** - serial number of the nesting card; "**length" - length of the** profile used for the nesting card; "**quantity" - number** of profiles equally **nested** on this card; "**rest_number**" - number of the **rest** used for the nesting card or 0, if the whole profile is used.

The "**plan**" element has the following additional elements: "**list_parts**" - **list of** parts located on the nesting map; "**rest**" - **rest** obtained on this nesting map.

The list of parts consists of one or more elements "**part**", which contains the attributes: "**number**" - part number from the job file; "**length" - length of the** part;

The residue has the following attributes: "**length**" - **length of the residue**; "quantity" - number of residues.

The number of residues coincides with the "**quantity**" attribute of the "plan" element - if two identical profiles are cut on the same map, then two identical residues are obtained.

Example of a result file.

```
<?xml version="1.0" encoding="windows-1251"?
<data>
<list_plans order_name="1003">
<plan number="1" length="1040" quantity="2" rest_number="1003">
<list_parts>.
  <part number="003494130002" length="447"/>
  <part number="003494150001" length="237"/>
```

```
</list_parts>.
<rest length="344" quantity="2"/>
</plan>.
<plan number="2" length="5600" quantity="1" rest_number="0">
<list_parts>.
  <part number="003494140001" length="1197"/>
  <part number="003494140001" length="1197"/>
  <part number="003494140001" length="1197"/>
  <part number="003494140001" length="1197"/>
</list_parts>.
<rest length="792" quantity="1"/>
</plan>.
</list_plans>.
</data>.
```

First of all, it was necessary to develop an algorithm to encode the solution of the problem as a gene sequence and synthesize the criteria and target function to evaluate chromosome adaptation.

It is worth noting here that it is not always obvious how to represent the solution of a particular problem in the form of a sequence of genes. In the classical formulation of GA [4], a chromosome consists of genes that can take values 0 or 1. Today, the rich experience of using GAs shows that a gene can take any value, and even represent entire data structures. In fact, a gene can be viewed not as a number, but as an object within the framework of the object-oriented approach. And the GA itself can be perceived as a general guideline - to present a solution in the form of a sequence of genes, mix them thoroughly and select the best sequence from the point of view of the target function. This is exactly such a variant and it is reasonable to use it when solving this problem.

At the beginning, the necessary data structures were developed.

Based on the analysis of the initial data and requirements to the solution, it became clear that the following basic structures will be needed to describe the profiles and parts (Fig. 2.). In the TProfile structure, in addition to the attributes specified in the initial data file, the NeedQuantity attribute is also required, which will store the number of profiles required for the solution of this problem.

Arrays TProfileArray and TDetaileArray require a separate explanation. Since the number of profiles and details are random, the nature of these arrays should be based on the concept of dynamic arrays. But since dynamic memory management is a very strict and sometimes complicated process, we are tempted to use ready-made solutions, for example, collections or vector class. vector is the same dynamic array, but it can manage the memory allocated to itself. Here we should take into account that the convenience of using ready-made solutions is paid for by their speed. In this case, further research has shown that the performance of the vector class is enough for an acceptable solution of a problem of maximum complexity.

Also, don't forget the overloading of the "==" and "<" operators, which allowed us to take advantage of the sorting functionality in the vector class.

The TGlobalData structure additionally contains the fields SumLen (total length of all parts), SumNotUsedWast (total length of unused scraps) and Unbroken (number of whole profiles used), which were later used to calculate the target function.

Further it was necessary to define what a gene is and how to encode the solution of the problem with its help. After a comprehensive analysis of the problem, it was accepted to consider one container, i.e., one profile, no matter whether it is a whole or a cut (Fig. 2), as a gene.

TProfile	
int	Type;
int	RestID;
int	Quantity;
double	Length;
int	NeedQuantity;

TDetail	
char	ID[13];
double	Length;
int	Quantity;

TProfilesList	
double	dRest_min_length;
double	dCutWidth;
TProfileArray	Array;

TDetailList	
int	OrderID;
TDetailArray	Array;

TGlobalData	
TProfilesList	Profiles;
TDetailList	Detailes;
double	SumLen;
double	SumNotUsedWast;
int	Unbroken;

Fig. 2. Basic structures of profiles and parts

TGen	
int	Type;
int	RestID;
double	Length;
double	Filed;
TDetailArray	DetailList;
int	Quantity;

TChromosome	
TGenArray	GenArray;
TTargetValue	TargetFunc;

Fig. 3. Structure of genes and chromosomes

The Filed attribute in the TGen structure represents the total length of all parts placed in this container, taking into account the cutting width. DetailList attribute represents the list of parts placed in the container. Quantity - the number of containers.

Thus, in our case a gene is an individual nesting map of one profile or its trimming. The union of the set of all trimmings and the set of all whole profiles is the chromosome blank for solving the problem. In the process of the algorithm work, the parts from the array TDetaileList (set of all parts) will be distributed to the array of containers TGenArray (set of all containers).

Obviously, it does not make sense to take into account all integer profiles. It is necessary to limit their number by solving the BPP problem by simple methods, for example, by a greedy algorithm.

It should be noted here that unlike classical GA, the chromosome length in our case is variable. We will have some initial set of containers, but as we solve the optimization problem, the size of this set will change. And it is impossible to predict in advance what will be the size of the ideal chromosome. We can get a solution based on two whole profiles, or a solution based on 9 scraps. And from the point of view of initial requirements, the second solution is preferable.

The next step was to develop a criterion for determining the target function of chromosome fitness. Here the main question was "how much better is one chromosome relative to another?".

Realizing that in the process of genetic algorithm operation, the total length of parts exceeding the size of the container will be placed in the containers, it was necessary to decide on the admissibility of such situations. On the one hand, it is possible to forbid to put into the container the parts that lead to overflow. On the other hand, there is a possibility that the part that caused the overflow is the most ideal fit for that particular container. Therefore, it is impractical to exclude such events. In order to quantitatively evaluate such variants, the criterion "overcrowding" was introduced, which shows by how much the total length of all parts placed in a given container, taking into account the width of the cut, exceeds the size of the container:

$$L_{ower} = N \cdot W_{cut} \cdot \sum_{i=1}^{N} L_d^i - L_p \qquad (1)$$

where W_{cut} is the width of the cut;

L_d^i - length of the *i-th* part in this container;

L_p - container size (profile length).

The ideal value of the criterion is $L_{ower} = 0$.

Next, a criterion was synthesized to evaluate the filling of the containers.

Consider a pair of identical containers that are half full. Consider also a second pair in which the containers are 0.6 and 0.4 full. On the one hand, the second option is more preferable. It utilizes the volume of the first container more fully and has more opportunities to use the second container. On the other hand, the amount of trimmings in both solutions is the same. Consequently, we need a criterion that allows us to distinguish between solutions that are identical from the point of view of the criteria specified in the problem condition. Since criterion (1) ideally tends to 0, the new criterion should also tend to 0 to simplify the search for the global extremum of the target function.

The criterion of "non-load factor" was introduced K_{load} which was initially defined as an S-shaped function:

$$K_{load} = 1 - e^{-\frac{(1-Filed)^2}{0.125}} \qquad (2)$$

where: Filed - value of the corresponding field of the TGen structure (Fig. 3).

The value K_{load} is an integral value that characterizes how fully loaded the containers are. 1 corresponds to an empty container, and 0 to a fully loaded container.

Further research showed that the criterion is not suitable in the form of (2).

First of all, it turned out that the fact of container overloading has a weak effect on the process of chromosome selection and there are quite a lot of such solutions in the general population. It was necessary to sharply "deteriorate" the adaptability of solutions in which overloading is present. The second point was that the same overloading could be obtained on different sets of

profiles. For example, on only whole profiles or on only trimmed profiles.

After a number of experiments, it was found that the best result can be obtained if we define the value of K_{load} as:

$$\begin{cases} K_{load} = \dfrac{\sum f}{N_b + N_u^0} \\ f = \begin{cases} 1 - e^{\frac{\left(\frac{L_p - Filed}{L_p}\right)^2}{0.125}}, \text{если } L_{ower} = 0 \\ 1, \text{если } L_{ower} > 0 \end{cases} \end{cases} \quad (3)$$

where: N_b - number of used residual profiles.

The next step is to synthesize the chromosome fitness function. The theory of multicriteria optimization offers many approaches to the formation of the target function. All methods have their own peculiarities, pros and cons. Within the framework of the solved problem we would like to discard solutions that have overloading without paying attention to the efficiency of packaging as a whole. Also the variant with less number of used whole profiles is more preferable in spite of everything else. Therefore, after a number of preliminary studies, the target was described by the following expression

$$J = 1000\frac{L_{ower}}{L_{sum}} + 100\frac{N_u^0}{N_{sum}^0} + 10K_{load} \quad (4)$$

where: L_{sum} is the total length of all parts;

N_u^0 - number of whole profiles used;

N_{sum}^0 - total number of whole profiles.

The ideal case is when there is no congestion and the first summand in (4) converges to 0.

The situation is similar with whole profiles. The ideal case is when the problem is solved using only cutoffs. Then the second summand in (4) also turns to 0.

Thus, we solved the problems of encoding the solution in the form of a chromosome and synthesized the chromosome adaptation evaluation function. Next, it was necessary to solve the most difficult part of the problem, namely, to synthesize the evolution algorithm.

For obvious reasons, the classical operations of crossing and chromosome mutation [4] are not applicable in our case. But we know that the task of these operations is to mix all the contents of genes as thoroughly as possible.

1) To reduce the dimensionality of the problem, we determine the number of required integer profiles without considering their number. No intelligent filling algorithms are used. If the next part does not fit into the given profile, we take the next one. Initialize the Error key. If the number of required profiles is greater than their real number, the problem may not have a solution Error = 1. Otherwise Error = 0.

2) Check whether the problem can be solved on the available set of profiles.

2.1) Form a sequence (chromosome) of containers consisting of existing scraps and real number of whole profiles. Sort the containers in ascending order.

2.2) If Error == 1 (C++ syntax. Two "=" signs correspond to a condition check as opposed to one corresponding to an assignment operation), we place all the parts sequentially in the first suitable container (taking into account its current filling). If all the parts cannot be placed, the problem has no solution. The available set of profiles is not sufficient. End of the algorithm.

3) We form the initial population of chromosomes.

3.1) Scatter the parts into containers in a random order. Form 10 such chromosomes.

3.2) We form the chromosome according to the principle of greedy algorithm. Starting from the longest part we pack them into the first suitable container.

4) Main Cycle.

4.1) If StaticCicle > *N* go to step 7. Exit condition: during *N* iterations the value of the target function of the best chromosome has not changed.

4.2) If the value of the target function of the best chromosome has not changed, increase the StaticCicle counter. If StaticCicle == *N* and there is overload in the best chromosome, then we reset the cycle counter to zero and continue searching for a solution for another *N* cycles. If the value of the target function of the best chromosome has changed at this step, then we reset the cycle counter to zero.

5) Cycle through the chromosomes of the current population

5.1) Remove empty integer profiles. This allows to further reduce the dimensionality of the problem.

5.2) Select a random non-empty container K_1. We take a random part from it and place it in another random container K_2, taking into account the width of the cut. We add the resulting solution to the next generation of the chromosome population. From the current chromosome we remove the containers K_1 and K_2.

5.3) Check whether container loading can be improved. If the container K_1 is not empty, go to section 5.4. otherwise go to section 5.7.

5.4) Cycle through the remaining containers of the chromosome.

5.5) If the total length of the parts of the container K_1 , taking into account the cutting width, is less than the size of the next container, taking into account its actual load, we transfer the parts from the container K_1 to it. We break the loop on the containers of the chromosome. We add the obtained solution to the next generation of the chromosome population.

5.6) If there is no suitable container K_1 among the remaining ones in the chromosome, we put its contents into a whole profile. We also put the contents of container K_2 into another whole profile. The resulting solution is added to the next generation of the chromosome population.

5.7) Repeat steps. 5.5-5.6 for the container K_2 .

5.8) Choose two containers at random and change their contents. The resulting solution is added to the next generation of the chromosome population.

5.9) Find the first overloaded container. Move a random part from it to any suitable container.

6) Calculate the value of the target function for all chromosomes. Sort them in ascending order. Perform the selection operation. We remove all less adapted chromosomes until the population size is reduced to 20 individuals. Return to step 4.

7) Form a file with the obtained solution in the required format.

Based on the results of this work, a dll-library was written and implemented at one of the furniture manufacturing enterprises. The computers available at the enterprise spent no more than 10 seconds to solve cutting tasks. The generated solutions fully satisfied the technologists. The number of stored scraps was minimized and the amount of unused waste was reduced.

Organization of input and output information in the form of xml-files allowed easy integration of the new software into the existing scheme of processing and transformation of information during design.

Conclusions

For all future researchers who will solve new problems using GA, I would like to give a couple of tips. GA is not intended for obtaining an ideal solution, but it allows you to obtain solutions close enough to optimal ones. Although by chance you can get an ideal solution. The dimensionality of the problem and the diversity in the generated populations have a significant impact on the running time. The algorithm and criteria presented above did not appear at once, but were the result of research, trial and error. It was the development of the algorithm for generating new generations that proved to be the most labor-intensive. On the one hand, it should maximize the variety of solutions obtained, and on the other hand, each new operation increases the running time. Therefore, we had to find a balance between the running time of the algorithm and the degree of acceptability of the solutions it produces.

References:

1. Valerio de Carvalho J.M. (2002). LP models for bin packing and cutting stock problems, European Journal of Operational Research, Volume 141, Issue 2, 2002, Pages 253-273, ISSN 0377-2217, https://doi.org/10.1016/S0377-2217(02)00124-8

2. Coffman E, Leung J, Csirik J. Variants of classical one-dimensional bin packing. Handbook of approximation algorithms and metaheuristics. Taylor & Francis; 2007 May. p. p. 33. DOI:10.1201/9781420010749.ch33

3. Delorme M., Iori M., Martello S. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms, European Journal of Operational Research, Volume 255, Issue 1, 2016, Pages 1-20, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2016.04.030.

4. Katoch S., Chauhan S.S., Kumar V. (2021) A review on genetic algorithm: past, present, and future. Multimed Tools Appl 80, 8091-8126. https://doi.org/10.1007/s11042-020-10139-6

РОЗВ'ЯЗАННЯ ЗАДАЧІ ОДНОМІРНОГО РОЗКРОЮ НА ОСНОВІ ГЕНЕТИЧНОГО АЛГОРИТМУ

Валентин Давидов, Ольга Тарахтій
Національний університет «Одеська політехніка»

Анотація. *У статті розглянуто процес синтезу алгоритму розв'язання задачі одновимірного розкрою меблевого профілю. Показано: процес проектування необхідних типів даних, вибір критеріїв та синтез цільової функції, синтез алгоритму пошуку рішення на основі елементів генетичного алгоритму.*

Ключові слова*: Задача одновимірного розкрою, генетичний алгоритм.*

Отримано 14.02.2024

Давидов Валентин Олегович, Національний університет «Одеська політехніка», доцент кафедри програмних і комп'ютерно-інтегрованих технологій, к.т.н., доц. Просп. Шевченка, 1, Одеса, Україна, E-mail: davydov@op.edu.ua. +38-096-787-63-31

Valentin Davydov, Odesa Polytechnic National University, Assistant professor of the Department of Software and Computer-Integration Technology, PhD, Assistant professor**.** Shevchenko ave., 1, Odessa, Ukraine
ORCID ID: 0000-0003-3099-7596

Тарахтій Ольга Сергіївна, Національний університет «Одеська політехніка», доцент кафедри програмних і комп'ютерно-інтегрованих технологій, к.т.н. Просп. Шевченка, 1, Одеса, Україна, E-mail: tarakhtij@op.edu.ua, тел. +38-067-587-40-71

Olga S. Tarakhtij, Odesa Polytechnic National University, Assistant professor of the Department of Software and Computer-Integration Technology, PhD. Shevchenko ave., 1, Odessa, Ukraine

ORCID ID: 0000-0002-4266-3481