Image Processing & Communication, vol. 17, no. 4, pp. 297-306
DOI: 10.2478/v10248-012-0058-3

EVOLUTIONARY ALGORITHMS APPROACH FOR CUTTING STOCK PROBLEM

ANDRZEJ ROMANOWSKI, ROBERT NOWOTNIAK, KACPER KAWECKI, TOMASZ JAWORSKI, ZBIGNIEW CHANIECKI, KRZYSZTOF GRUDZIEŃ

Institute of Applied Computer Science, Lodz University of Technology, Poland
androm@kis.p.lodz.pl

Abstract. This paper contain study of three algorithms for optimisation of use of materials for cutting process. Cutting Stock Problem (CSP) and one dimensional guillotine cat variant of the CSP is introduced. Afterwards three different way of solving the problem are presented. For each of theme one algorithm is proposed. First is creating all the possible solutions and choosing the best one. Second is trying to recreate a human thinking process by using a heuristic search. Third one is inspired by an evolution process in the nature. Design and implementation of each of them is presented. Proposed algorithms are tested and compared to each other and also to the other known solutions.

1 Introduction

1.1 Background

Optimization of material use in cutting processes is a complex problem. The issue of optimal cutting occurs in many areas of industry - hard board, wood, paper, window, metal, glass and other industries. For each one, the problem is a bit different, also each one have a different optimality criteria, but there are some which are common for all of them. Problem could be defined in one, two or three-dimensional space, cuts could be made in a guillotine or non-guillotine style, and many other cases.

1.2 Definition of Cutting Stock Problem

General definition of Cutting Stock Problem (CSP) is " *Smaller pieces with a given order demand have to be cut from larger stock material. The objective is to minimize the amount of stock material needed to produce the ordered pieces*" [1]. In real life, there are plenty of CSP variants. They mainly differ in additional restrictions, which real occurrence of it has different applications. In a wood industry, the thickness of a blade needs to be considered, when in a paper industry, often blade thickness is negligible. When pipes are going to cut, only one dimension is taken into account, while for cutting hardboards two dimensions must be considered. There are many more aspects and restriction which can be thought of.

One-dimensional guillotine cut has many applications, mostly in pipe, paper, steel and fibre industries. It occurs when we consider only one dimension (length), and a stock material all the time is cut in the same way -

mostly in perpendicular to the considered dimension. If all stock materials have the same length, it can be examined as one-dimensional cutting stock problem (1D-CSP). Main costs of 1D-CSP are remaining pieces after cut process, called trim-loss, which are in most cases treated as a waste of a stock material. Problem was studied with regard to minimization of trim-loss and reduction of a number of different cutting patterns. Constraint of equal size of the stock material simplifies the problem. Removing constraint makes the issue more complicated. Considered case is that each stock material can have a different length. In a real life, such case occurs commonly in the wood industry, but often in different industries there is more then one length of the stock material available.

1.3 Quality of solutions vs. time complexity of different algorithms

In most of analyses of the CSP problem, the main issue is cost of material or cost of cutting, but a time cost of finding the best solution is ignored. The cost of finding the optimal solution is an important economical aspect. Imagine that someone can spend lot of resources to find the optimal solution and only some small part of it is obtained from its result. For example, there is machinery, which can cut 10 pieces in one minute. Average order contains about 250 different pieces. Cutting each order take about 25 minutes. So solution for next order should be found in this time. If problem is very complicated finding optimal solution can take much more time. In case when calculation is much longer then this 25 minutes, probably less expensive will be using one or two stock materials more, then stopping machinery. Therefore, a suboptimal solution can be treated as proper one, as long as it is available in remarkably shorter time. This paper presents that is it possible to find a satisfying suboptimal result within satisfactory short time.

2 Selected algorithms

CSP is a well-known problem, and a lot of work can be found in literature. However, there are still areas for novel approaches and improvement, since there is a plethora of criteria that can be posed on the problem solution. There are three algorithms proposed in this paper to be somehow face each other since they are based on different approaches. First one is to obtain the optimal solution to be compared with evolutionary-based other two.

2.1 Brute-force search

The first idea, perfect for a benchmark for the other approaches, is to find all possible combination of the cutting stock material into the pieces and choose the best one in terms of optimal material use. Brute-force search or exhaustive search is a very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. This can be accomplished using different algorithms, however tree search seems to be the most common choice for Cutting Stock Problem. In the case when all the required output pieces have to be the same length, the solution is trivial. However, for different lengths of pieces, the sorted list of pieces needs to be prepared. As the order of different length of pieces may influence the solution, all possible combinations of order need to be verified. Therefore, Breadth-first search for checking all the sequences, and for each node of the tree run the Depth-first search for finding the optimal materials sequence can be applied. Such solution can find optimal solution [2].

2.2 Heuristic algorithm (human thinking based)

The solution proposed in 2.2 is quite complex and need lot of memory for complicated problems. Searching in such tree needs a lot of resources. Dealing with such problems needs some intelligent mechanisms. Artificial intelligence

often tries to simulate a human thinking. For human brain, it is easy to find a right combination of two pieces, but for a bigger amount is much harder; hence an algorithm is required to proceed with these steps. Since this operation is similar to the route-finding problem, the solution such as heuristic search strategy can be adopted here. There is a big family of such algorithms. Main difference between them is an evaluation function, which in special case is called heuristic function $h(n)$ According to [3] heuristic function is the estimated cost of the cheapest path form node n to goal. The A* (A-star) algorithm uses also $g(n)$ which is the cost of reaching the node. The optimality of solution is proved here [3, 4]. The real and important cost in the CSP is a material waste, so the heuristic function should predict the material waste, however it is difficult to predict. Fortunately prediction doesn't have to be very precise. It should help to choose the best node. Wrong prediction doesn't mean that the algorithm won't find the solution, but proper prediction would accelerate the algorithm [10]. The proposition of the function which will predict the cost is using a modulo operation. If $h(n)$ is an estimated distance and $\min(p)$ returns length of the smallest available piece the proposed function can be described as (1):

$$g(n) = h(n) \ \mathbf{mod} \ \min(p) \tag{1}$$

prediction could be very imprecise for $h(n)$ significantly greater then $\min(p)$. However. (2):

$$k \ \mathbf{mod} \ n < n \tag{2}$$

In this case $h(n)$ is also significantly greater then $g(n)$. It means that: firstly algorithm will choose pieces due to the length (longest pieces first), but when it comes to nodes which are close to the solution the cost factor become important. Such solution quite accurately recreates human thinking.

2.3 Nature-inspired algorithm (genetic algorithm)

Genetic algorithms (GAs) were created by John Holland, from University of Michigan in 1960s. They were invented to study a phenomenon of an adaptation in the nature [4, 5]. Years later they were adapted to solve other problems. In genetic algorithms process of genetic reproduction is imitated including gene recombination and gene mutation. There is no rule how many times the algorithm should be iterated. The interesting fact is, that it is barely feasible to obtain the same population in each run of the algorithm. It is due to the fact, that randomness plays an important role there. Genetic algorithms with replaced the binary representation are called evolution algorithms. There is no limitation for genetic operators, while in GAs there are only crossover and mutation operators. Operators are divided in two groups one or multi-arguments. Example of the one-argument operator is mutation. In evolution algorithms there could be few different operators for mutations and crossings.

Authors considered all constraints, and decided that the easiest way will be using the so-called evolution strategy. The main difference from the genetic and evolution algorithm is, that there is no population of one chromosome, but set of different chromosomes creating one specimen, whereas proposed is built from the chromosomes. Each of the chromosomes represents one particular stock material. Each chromosome will contain a set of genes. Each gene represents one particular pieces or an empty space (none). Algorithm will run in the following way. Create specimen with set of chromosomes containing empty genes. Randomly set genes for chromosomes in such way, that one gene can be used only once, and all gens are used. Rate the whole specimen. Remember the mark. Rate chromosomes. Select pairs of random chromosomes. Make some crossing between chromosomes. Make some mutation. Rate whole specimen, if rate is better then previous, remember specimen. Repeat steps from rating each

of chromosomes. Crossing and mutation operation must be done in such way, that no piece is changed or taken out from the solution. Proposed solution is to make a crossing between different chromosomes, which is far from natural mechanisms. However, for the presented problem of interest this arrangement should work fine. Mutation operator could make random swap of single genes. Moreover to obtain good result, only crossovers that make results better should be applied. It is due to the fact, that there is no population created. When there is population of chromosomes, there is also mechanism of natural selection, which eliminates week specimens. If it comes to mutation process, it can't be limited only to the mutation, which make result betters. When the algorithm will get stuck in some local maximum, it is hardly probable that cross operators can get out from it. For getting out of it the mutation operator are used [5, 6]. Proposed algorithm is far from classical genetic algorithms, it is even far of the evolution algorithm. However, it is based on the same idea - the natural mechanism of evolution. More theory and related work can be found here [10, 11, 12, 13, 14].

3 Numerical experiments

The algorithms have been compared with three different ways. The first stage was finding the difference in a result due to parameters, which can be set. The next stage was comparing created algorithms to each other. The last stage was comparing algorithm with results from other research. For testing, the examples from [6] and [7] have been drawn.

3.1 Comparison criteria

Most important criterion is the optimality. It will be shown in the percentage of use of the material, or in the percentage of the material waste (trim losses). However, comparing only the optimality without any relation of the results it will be only in last stage of tests. The time of the calculation is a highly important factor for the end-user. However, it is strictly dependent on an implementation, so comparing due to the time of the calculation is not objective. Therefore, optimality of solution will be related to the time of calculation only.

3.2 Datasets

Dataset 1 has been taken from [6]. Originally, it has an unlimited number of material pieces of 1900mm in length each piece. For testing purpose, we used 15 - according to [6] for the optimal solution only 8 is needed. The required piece list is the following:

- 8 pieces 330 mm each
- 8 pieces 360 mm each
- 13 pieces 385 mm each
- 11 pieces 415 mm each.

Dataset 2 is taken from [7]. Originally it has an unlimited number of material pieces of 1900mm in length each piece. For testing purpose we used 18 - according to [6] for the optimal solution only 13 is needed. The required piece list is the following:

- 8 pieces 340 mm each
- 8 pieces 365 mm each
- 13 pieces 385 mm each
- 11 pieces 415 mm each
- 5 pieces 435 mm each
- 6 pieces 260 mm each
- 4 pieces 300 mm each
- 7 pieces 320 mm each
- 3 pieces 335 mm each.

Dataset 3 is taken from [7]. Originally it has an unlimited number of two types of material pieces of 1900mm and 2200mm in length accordingly. For testing purpose we used 10 of each type - according to [7] for the optimal solution only 6 of each type is needed. The required piece list is the following:

- 9 pieces 340 mm each
- 8 pieces 365 mm each
- 13 pieces 385 mm each
- 11 pieces 415 mm each
- 5 pieces 435 mm each
- 6 pieces 260 mm each
- 4 pieces 300 mm each
- 7 pieces 320 mm each
- 3 pieces 335 mm each.

Dataset 4 is taken from [7]. Originally it has an unlimited number of material pieces of 5600mm in length. For testing purpose we used 80 of each type - according to [8] for the optimal solution only 73 is required. The required piece list is the following:

- 22 pieces 1380 mm each
- 25 pieces 1520 mm each
- 12 pieces 1560 mm each
- 14 pieces 1710 mm each
- 18 pieces 1820 mm each
- 18 pieces 1880 mm each
- 20 pieces 1930 mm each
- 10 pieces 2000 mm each
- 12 pieces 2050 mm each
- 14 pieces 2100 mm each
- 16 pieces 2140 mm each
- 18 pieces 2150 mm each
- 20 pieces 2200 mm each.

4 Results

4.1 Parameter comparison

In this test, only Heuristic Algorithm, and Genetic Algorithms have been compared. This is due to the fact, that the Tree Search does not have any parameters, which can have an influence on the results. The first one implemented has one parameter that can be adjusted. It is a value telling how good solution should be treated as satisfyingly optimal. In the algorithm class this value is kept as a floating-point number, which should fall between 0.0 and 1.0. Setting allows 10 different value of the parameter from 0.900 up to 0.999 with step 0.011.

However, if the value is lower then a total length of pieces divided by a total length of materials (minimal optimality needed in order to find the solution), it is changed to this coefficient. For the tests where only one length of the stock material is used, each run has exactly the same results, but when there is more length of material, there is some randomness, so the test run 10 times and the results are averaged. Results are shown in Tab. 1-4.

One can observe on dataset 1,2,3 that the heuristic mechanism give us a quite good estimation of the result. It can observed that it gives some limit, under it, there is no difference what parameter is set. Above this limit, parameters have a significant influence on time of calculation. Also it changes an amount of the trim losses. On dataset 4 it can be observed that a higher value of the parameter not always results in better solution.

Interesting think is in results of dataset 3. For a proper analysis Tab. 5 must be taken into a consideration. If only the minimum values of function calls are examine, result is similar to rest of the tests. Why average and maximum values differ so much? It is because the algorithm selects random order of stock material. As result shows, these orders have big influence on the algorithm performance.

Taking in to consideration that for one function execution is more then one random factor, and there is about

Tab. 1: Results for the parameter test of Heuristic Algorithm for the dataset 1

parameter	material used	waste	function calls	time
0.900	9	7.1%	16 882	0.129 s
0.911	9	7.1%	16 882	0.136 s
0.922	9	7.1%	16 882	0.139 s
0.933	9	7.1%	16 882	0.135 s
0.944	9	7.1%	16 882	0.134 s
0.955	9	7.1%	16 882	0.135 s
0.966	9	7.1%	16 882	0.139 s
0.977	9	7.1%	16 882	0.138 s
0.988	8	0.4%	511 760	0.971 s
0.999	8	0.4%	74 671 228	314.859 s

Tab. 2: Results for the parameter test of Human Thinking Based Algorithm for the dataset 2

parameter	material used	waste	function calls	time
0.900	13	3.94%	8 503	0.088 s
0.911	13	3.94%	8 503	0.090 s
0.922	13	3.94%	8 503	0.085 s
0.933	13	3.94%	56 758	0.218 s
0.944	13	3.94%	62 158	0.290 s
0.955	13	3.94%	179 593	1.219 s
0.966	13	3.94%	179 593	1.288 s
0.977	13	3.94%	1 067 238	3.823 s
0.988	13	3.94%	25 990 200	117.518 s
0.999	13	?	>617 426 704	> 10 h

Tab. 3: Results for the parameter test of Human Thinking Based Algorithm for the dataset 3

parameter	material 1 used	material 2 used	waste	function calls	time
0.900	6.3	6.0	3.52%	8 521	0.109 s
0.911	6.4	5.7	2.37%	10 629	0.141 s
0.922	5.6	6.4	2.42%	18 672	0.161 s
0.933	5.9	6.2	2.66%	160 190	1.010 s
0.944	6.1	5.9	2.20%	135 944	0.882 s
0.955	6.2	5.8	1.98%	67 189	0.504 s
0.966	5.7	6.3	2.35%	176 888	1.373 s
0.977	6.0	6.0	2.13%	136 931	0.841 s
0.988	6.8	5.3	1.93%	4 739 013	20.231 s
0.999	?	?	?	?	?

Tab. 4: Results for the parameter test of Human Thinking Based Algorithm for the dataset 4

parameter	material used	waste	function calls	time
0.900	75	2.55%	44 562	0.464 s
0.911	75	2.55%	44 562	0.491 s
0.922	75	2.55%	44 562	0.452 s
0.933	75	2.55%	44 562	0.464 s
0.944	75	2.55%	44 562	0.450 s
0.955	75	2.55%	44 562	0.500 s
0.966	75	2.55%	44 562	0.505 s
0.977	76	3.66%	1 553 681	4.993 s
0.988	74	1.44%	8 318 912	19.459 s
0.999	74	1.44%	3 364 290 321	5316.027 s

Tab. 5: Diversity function calls for Dataset 3

parameter	average	maximum	minimum
0.900	8 521	18 184	7 078
0.911	10 629	26 662	7 102
0.922	18 672	93 478	7 359
0.933	160 190	779 828	7 260
0.944	135 944	700 556	7 359
0.955	67 189	316 902	7 365
0.966	176 888	1 441 111	7 221
0.977	136 931	376 658	7 476
0.988	4 739 013	16 545 469	93 911

Tab. 6: Cross probability parameter comparison for the dataset 1

parameter	iteration	materials used	function calls	time	function calls per iteration	function calls per second
0.25	5 325	9.3	178 526	3.993	34.121	43 428
0.30	5 923	9.1	205 270	4.366	34.926	46 234
0.35	5 283	9.1	189 363	3.930	36.170	46 779
0.40	4 057	9.1	150 623	3.061	37.532	48 288
0.45	4 504	9.1	172 602	3.714	38.742	45 348
0.50	5 038	9.1	199 466	4.099	39.860	49 726
0.55	4 374	9.0	178 122	3.622	40.972	48 150
0.60	4 135	9.1	178 286	3.620	44.743	49 396
0.65	2 588	9.0	112 290	2.258	43.739	49 115
0.70	4 110	9.0	183 160	3.901	44.552	47 962
0.75	3 178	9.0	144 166	2.744	45.554	51 811

one million function calls, average value taken from 10 test cannot give us a proper statistic. However calculation of the rate between the time, function calls or iteration give us rather stable values. With growth of the parameter value the ration of function calls per iteration is recognizably growing. This mean there is more cross operation done, and each operation cause more function calls. Also amount of function calls per second is growing, but the growth is not so stable - it is also dependent from random events. The same we can observe with trim loss. The relation of mutation parameter comparison for the separate datasets show interesting features, which can be seen if plotted. For dataset 1 the difference between parameter 0.025 and 0.075 is not significant, but it can be derived that bigger probability of the mutation causes more drops. It can be more clearly observed for dataset 4. In these places, mutation made change that lowered the rate. Such changes are useful when the algorithm get stacked in some local maximum. However, when there are too many mutations, which have a negative influence on rate, it could slow down the algorithm, or even make a right solution impossible to find. When there are too less mutations, it makes the algorithm stacked in local maximums for long time, so in fact it also slows down the algorithm.

4.2 Performance comparison with existing implementations

Dataset 1: Described optimal solution need 8 stock material and it gives average trim loss 0.724%. Both algorithms returned a solution with 9 stock materials when parameters were set to default values. However when human thinking based algorithm has its parameter set to maximum there is an optimal solution found. Also, nature mechanism based algorithm can find such solution but with the given input it took a long time and it need luck. Optimal solution was found in 2 runs out of 100. However, when the amount of the available stock material has been changed to 8, the optimal solution was always found.

Dataset 2: Optimal solution for this dataset is when 13 materials is used. Such result is obtained without any problem by human thinking based algorithm. Probability of obtaining this solution by nature mechanisms based algorithm with default parameters is around 7%.

Dataset 3; Due to the [7], for the optimal solution 6 materials of both lengths (1900mm and 2200mm) are needed with the average trim losses 3.466%. However, the solution does not care about the constraint of amount of cuts. Human thinking based algorithm found solution containing of 1 materials of the length 1900mm and 10 materials 2200mm long, which results in the average trim loss equal to 0.659%. The genetic algorithm seems to be stuck in a local maximum and it gives result containing 10 materials of the length 1900mm and 3 materials of the length 2200mm, which result in 4,561% of the trim loss. Again, setting the available materials to the minimal needed amount result in the optimal solution. Dataset 4: According to [8] 19 different optimal solution exists, for which 73 stock material are needed resulting in 0.401% of waste. Human thinking based algorithm with default parameters return solution with 75 raw materials. Setting the parameter to the maximum results in 74 materials solution - it cannot find the optimal solution. Nature mechanisms based algorithm with the default values return result containing 79 materials, by changing it can be reduced up to 75 materials. It is disputable if assumed by [8] solutions exist. We set up available materials to 73 and run nature mechanisms based algorithm. Proper solution was found 1 time per 20 runs.

5 Final conclusions

In this paper, three different algorithms have been considered. All of them work and they find an optimal or suboptimal solution. Brute-force algorithm is rather useless in a real life due to its time complexity. However, it gave a comparison, and shows how significantly heuristics and

artificial intelligence can improve the performance. The two other algorithms work fast. The heuristic algorithm was able to find an optimal solution for 3 from 4 datasets, and for the last datasets it was able to find a suboptimal solution, which was near from an optimal one. The algorithm presents how good can be recreating human thinking.

The genetic algorithm for each dataset was finding rather good suboptimal solution, but when the available stock material was limited to minimum needed in most cases, it was finding an optimal solution, even when heuristic algorithm was unable to find such one. Probably further research and development on this algorithm can result in finding an optimal solution in most cases. The both proposed algorithms (heuristic and genetic) can be used in the industry; nevertheless the heuristic algorithm has greater chance because it gives more stable results. The presented genetic algorithm would need some improvements before it works satisfyingly well. Both algorithms are a good alternative for known exhaustive way of solving one-dimensional Cutting Stock Problem.

References

[1] C. Nitsche, G. Scheithauer, J. Terno, Tighter relaxations for the cutting stock problem, *European Journal of Operational Research*, No. 112, pp. 654-663, 1999

[2] T.H. Cormen , C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, Second Edition, The Massachusetts Institute of Technology, 2001

[3] S.J. Russel, P. Norvig, *Artificial Intelligence, A Modern Approach*, Second Edition, Pearson Education, 2003

[4] M. Melanie, *An Introduction to Genetic Algorithms*, The Massachusetts Institute of Technology, 1996

[5] B.J. Wagner, A genetic algorithm solution for one-dimensional bundled stock cutting, *European Journal of Operational Research*, No. 117, pp. 368-381, 1999

[6] G. Schilling, M.C. Georgiadis, An algorithm for the determination of optimal cutting patterns, *Computers & Operations Research*, No. 29, pp. 1041-1058, 2002

[7] CSP - cutting stock problem, http://en.wikipedia.org/wiki/ Cutting_stock_problem, 2010 (retrieved as of may 2012)

[8] R. Morabito, L. Belluzzo, Optimising the cutting of wood fibre plates in the hardboard industry, *European Journal of Operational Research*, No. 183, pp. 1405-1420, 2007

[9] S. Umetani, M. Yagiura, T. Ibaraki, One-dimensional cutting stock problem to minimize the number of different patterns, *European Journal of Operational Research*, No. 146, pp. 388-402, 2003

[10] E. Hopper, B. Turton, A genetic algorithm for a 2d industrial packing problem, *Computers & Industrial Engineering*, No. 37, pp. 375-378, 1999

[11] N. Siu, E. Elghoneimy, Y. Wang, W. Gruver, M. Fleetwood, D. Ko- tak, Rough mill component scheduling: Heuristic search versus genetic algorithms, *IEEE International Conference on Systems, Man and Cybernetics*, pp. 4226-4231, 2004

[12] R. Alvarez-Valdes, A. Parajon, J.M. Tamarit, A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems, *Computers & Operations Research*, No. 29, pp. 925-947, 2002

[13] M. Hi, Exact algorithms for unconstrained three-dimensional cutting problems: a comparative study,