An Enhanced Genetic Algorithm with the BLF2G Guillotine Placement Heuristic for the Orthogonal Cutting-Stock Problem*

Slimane Abou-Msabah[1] — Ahmed Riadh Baba-Ali[2]— Basma Sagr[1]

[1] Department of Computer Science, University of Science and Technology Houari Boumedienne, USTHB, Bab Ezzouar, Algiers, Algeria, slmalg@yahoo.com
[2] Department of Electronics, University of Science and Technology Houari Boumedienne, USTHB, Bab Ezzouar, Algiers, Algeria, riadhbabaali@yahoo.fr

Abstract. The orthogonal cutting-stock problem tries to place a given set of items into a minimum number of identically sized bins. As a part of solving this problem with the guillotine constraint, the authors propose combining the new BLF2G, Bottom Left Fill 2 direction Guillotine, placement heuristic with an advanced genetic algorithm. According to the item order, the BLF2G heuristic creates a direct placement of items in bins to give a cutting format. The genetic algorithm exploits the search space to find the supposed optimal item order. Other methods try to guide the evolutionary process by introducing a greedy heuristic to the initial population to enhance the results. The authors propose enriching the population via qualified individuals, without disturbing the genetic phase, by introducing a new enhancement to guide the evolutionary process. The evolution of the GA process is controlled, and when no improvements after some number of iterations are observed, a qualified individual is injected to the population to avoid premature convergence to a local optimum. To enrich the evolutionary process with qualified chromosomes a set of order-based individuals are generated. Our method is compared with other heuristics and metaheuristics found in the literature on existing data sets.

KEYWORDS: Cutting and Packing, Guillotine Constraint, Combinatorial optimization, Genetic algorithms, Heuristics, premature convergence, local optimum.

1. Introduction

The cutting or packing problem is a combinatorial optimization problem. The objective is to determine a suitable arrangement of various items within a wider set of bins. The main objective is to maximize the use of raw materials, thus minimizing losses. This problem is interesting because it is applicable to several fields. For example, in the wood or steel industries, it is necessary to consider how to cut rectangular pieces from large sheets of material. In the transportation and logistics fields, objects of different sizes have to be packed in larger containers of standard size. In floor planning, it is

* paper under review

necessary to consider very-large-scale integration (VLSI) design. If a cost equal to its area is assigned to each piece, this problem can be formulated as a knapsack problem.

This paper considers the orthogonal cutting-stock problem, which utilizes a strip of fixed width and supposed infinite height to generate items of rectangular shape. Since items are packed in levels; with height equal to the height of the tallest item in the level; these generated levels are projected directly to bins, so our aim is to reduce the height of levels in the strip. The production machines can be the guillotine shears, which impose the cut from edge-to-edge (the guillotine constraint). The items keep their original orientations to be cut in decorated plates or for the draft layout of pages of newspapers (the orientation constraint).

Several placement heuristics are used to solve this problem. The authors find that the new BLF2G guillotine placement heuristic introduced by Msabah and Baba-Ali [1] is the most adaptive heuristic, since it packs items in levels to ensure that the guillotine constraint is satisfied and has a strong policy to exploit gaps vertically and horizontally by checking the guillotine constraint vs FC and SHF methods.

In this paper, we are going to combine this heuristic with a genetic algorithm improved cleverly. According to the item order, the BLF2G heuristic makes a direct placement of items on levels to give a cutting format. The genetic algorithm exploits the search space to find a supposed optimal order. We introduce a notion of the stability of the evolutionary process, i.e., we observe that, when there are no improvements, a locally optimal solution has been found. After stability is detected, we propose injecting an ordered list of items into the population based on a greedy heuristic to diversify the search in another area of the solution space. We also propose a set of order-based heuristics to be injected into the population to enhance the ability of the genetic algorithm to find good solutions.

After this introduction to the problem, we will discuss in section 2 the guillotine placement heuristics found in the literature that verify the guillotine constraint adapted to our case. In section 3, we present the genetic algorithm and propose a set of fast greedy techniques that gives qualified order-based individuals. The computational experiments will be discussed in section 4, followed by our improvement where we describe the stability controlled genetic algorithm, and then we make a comparison of our method to other heuristics on data sets found in the literature. We shall end our article with the conclusion and further work, which are presented in section 5.

2. Guillotine placement heuristics

In this section, we are interested in investigating placement heuristics from existing methods found in the literature to propose a heuristic that fits our case. We focused our research on guillotinables heuristics, which are suitable for an edge-to-edge cut.

We first consider the Floor-Ceiling (FC) approach introduced by Lodi et al. [2], which separates the placement of items into two levels. The FC approach places the items from left to right at the bottom of the level (floor). When no more items will fit on the current level, the approach attempts to place items from right to left at the top of the level (ceiling). They propose a new variant of the FC approach to check the guillotine constraint which performs the cuttings from edge to edge, bold lines, as shown in Figure1

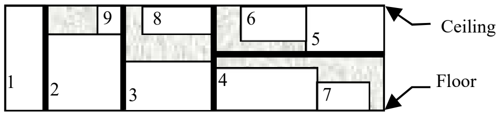

Fig. 1. The guillotine variant of the FC algorithm proposed by Lodi et al. [2]

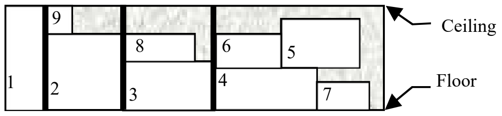

Fig. 2. The SHF algorithm proposed by Ben Messaoud et al. [3]

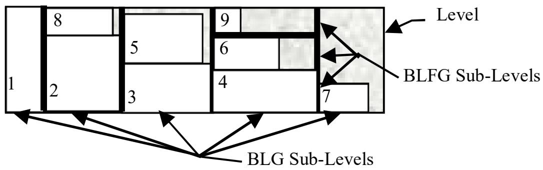

Fig. 3. The BLF2G guillotine placement heuristic [1]

Ben Messaoud et al. [3] modified the guillotine variant of the FC approach and proposed a Shelf Heuristic Filling approach (SHF). They propose to inject items placed in the ceiling from right to left below from left to right, as shown in Figure 2.

Recently, Msabah and Baba-Ali [1] proposed a new guillotine placement approach based on levels and proposed exploiting intra-level residues while checking the guillotine constraint. An item can be laid out on the strip in three possible ways: Fig. 3.

Placement in levels: The strip is structured in levels, and the items are packed according to the famous BL heuristic, that places items sequentially at the first Bottom Left suitable position. When no more space is available in the current level they create a new level, and so on. For each item packed horizontally in a level a Bottom Left Guillotinable Sub-Level “BLGSub-Level” is created.

Placement in BLGSub-level: Items are placed in a vertical order on these residues; which are delimited by the width of the bottom item and the height of the level; according to the Bottom Left heuristic. For each item packed vertically in a BLGSub-Level a Bottom Left Fill Guillotinable Sub-Level “BLFGSub-Level” is created.

Placement in BLFGSub-level: Items are placed in BLFGSub-levels horizontally.

3. Our contribution

The BLF2G guillotine placement heuristic is the most adaptable to our case among the studied approaches, because it verifies the guillotine constraint, and has a strong policy to exploit gaps. We will combine it with a genetic algorithm. According to the order of items, in a given individual, the BLF2G heuristic packs items directly on a strip to give a layout.

The genetic algorithm investigates the solution space to find the best order of items that offer good results by applying the BLF2G placement heuristic. An initial randomly chosen population is created. All the individuals in this population are evaluated by applying the BLF2G heuristic; to each individual we assign a fitness corresponding to the height of the items packed in the strip. Depending in the fitness and randomly chosen operators, the genetic algorithm evolves till stop criteria achieved or the optimal solution is reached, i.e. when the sum of the surfaces of all items is equal to the surface of the strip.

As introduced in Msabah and Baba-Ali [1], the genetic algorithm failed vs. a greedy heuristic for a large data set; sometimes the greedy heuristic gives an immediate good result with regard to the GA which requires much more time to give a less effective result. They propose to guide the genetic algorithm by introducing sorted lists of items to the initial population; they called this approach BLF2G+GA$_{imprv}$. There are several sorting policies. We will propose some new sorting policies that promote characterized items to appear first in the layout process. We use these greedy policies, to be injected in the initial population, to enhance the quality of the GA, and we will surmount various difficulties, which will be discussed below.

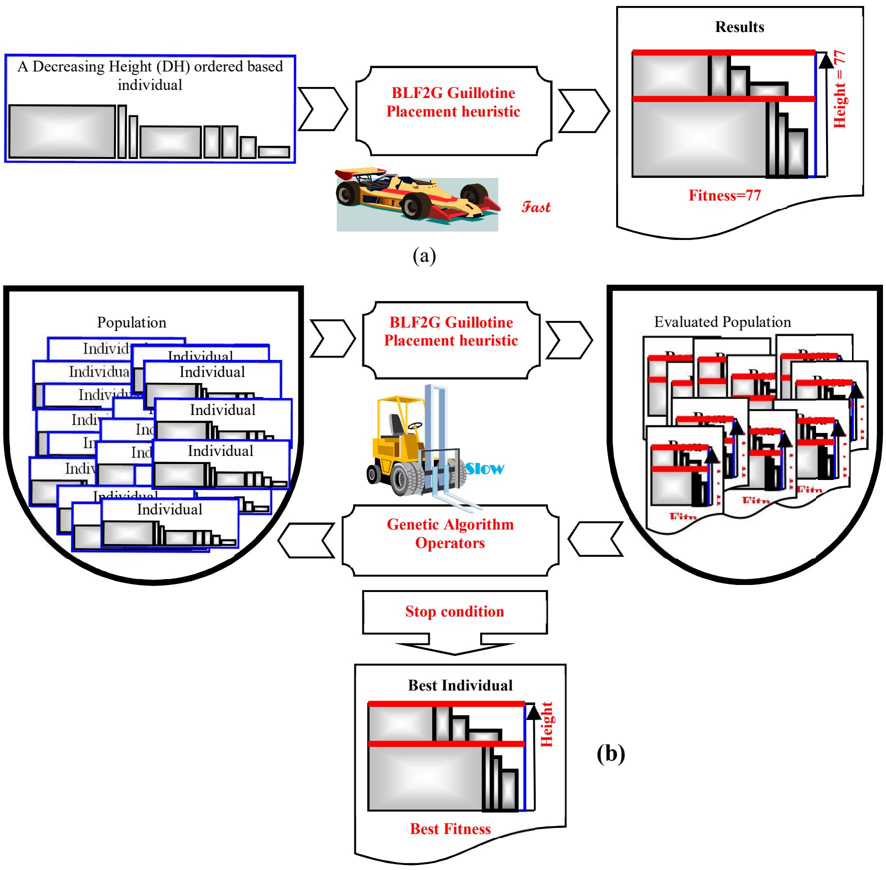

Fig. 4. (a) a fast greedy heuristic and (b) a slow standard genetic algorithm

A greedy heuristic, such as the well-known Decreasing Height (DH); which sorts out items according to the decreasing height policy; gives an instant result in around one second depending on the problem size, and it is efficient for large sized problems. A standard GA method is less efficient and slow for large sized problems but gives better results for small sized problems. (cf. [1]).

3.1. **The genetic algorithm**

We used a real-coded genome (cf. [5]). Each item has an identifier; the chromosome is defined as being a suite of identifiers, which determines the order of appearance of items in the chromosome. Based on the BLF2G policy, the appearance order of items in the layout process determined the quality of every individual. We implemented a genetic algorithm approach with a population size of 100, and we fixed the number of generations to 20 times the number of items. Initially, we generated a random population with a random ordering item in each individual. At each generation, our BLF2G policy gave the quality of each individual. The genetic operators were defined as follows:

Crossover operator: The crossover operator used is based on one-point cut operator (cf. [9]). The crossover rate is 0.8. The application of the crossover gives an invalid offspring; we made a correction to make the children valid. We corrected child 1 by replacing the double genes by the missing genes according to their order of appearance in parent 2 and replaced the double genes in child 2 by the missing genes according to their order of appearance in parent 1.

Mutation operator: This is about swapping two randomly chosen sites at which the mutation rate is 0.15, figure 5.

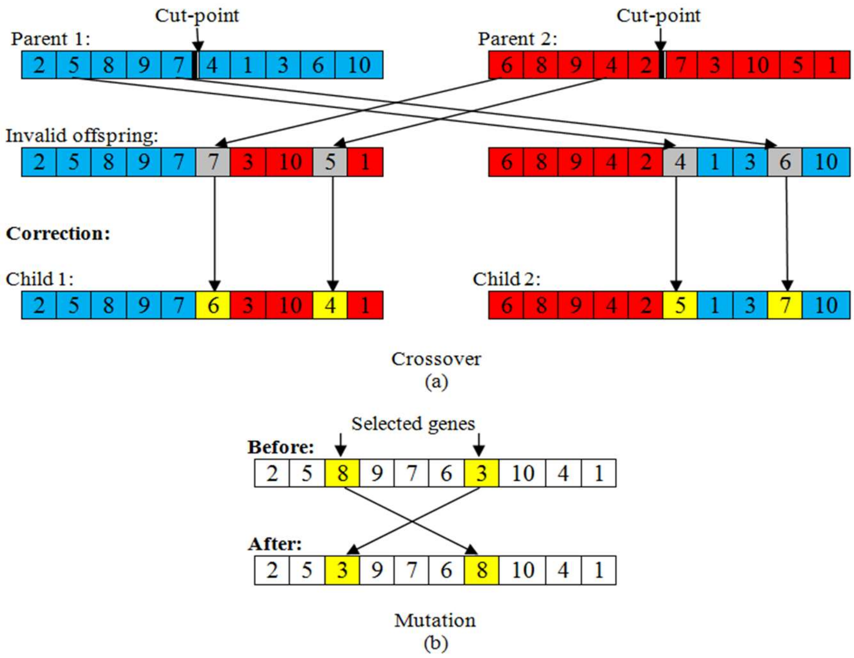

Fig. 5. The genetic operators, crossover and mutation.

3.2. **Fast greedy heuristics**

In this section, the authors will present various fast order-based policies, ranging from a simple greedy policy to a more sophisticated one, to be injected in the evolutionary process. The objective is to generate qualified individuals by applying fast greedy techniques to be integrated in the evolutionary process, Fig. 6, fig. 7.

- Decreasing Height (DH) policy: The list of items is sorted according to the decreasing height of items. When two items have the same height, we promote the widest one; see Figure 6 (a).
- Increasing Height (IH) policy: The list of items is sorted according to the increasing height of items. When two items have the same height, we promote the smallest one; see Figure 6 (b).
- DH-Reverse policy: This policy applies the DH heuristic on two sides. We put the longest item on the right side, then the next longest one on the left side, and so on, until the last item, and then we concatenate the right side with the inversed list of the left side to obtain a DH-Reverse sorted list; see Fig. 6 (c).
- IH-Reverse policy: This policy applies the IH heuristic on two sides. We put the shortest item on the right side, then the next shortest one on the left side, and so on, until the last item, and then we concatenate the right side with the inversed list of the left side to obtain an IH-Reverse sorted list; see Fig. 6 (d).
- Harmonic policy: This policy applies, alternately, the IH heuristic and then the DH heuristic. The list of items is sorted alternately: take the highest item, then the shortest one, and so on; see Fig. 6 (e).

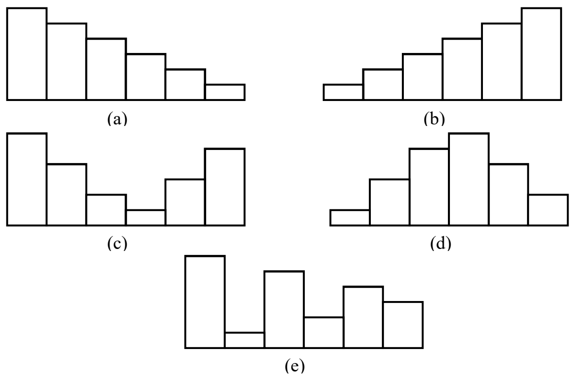

Fig. 6. The greedy heuristics, items sorted according to different sorting policies

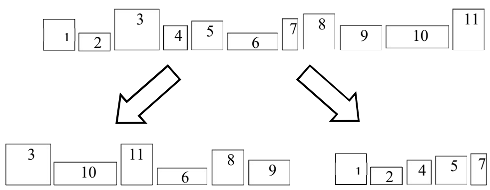

Fig. 7. The Divide Rule policy

- Decreasing Height Optimization Width heuristic (DHOptW): Msabah and Baba-Ali [1] proposed this intelligently sorted policy by simulating the BLF2G guillotine placement heuristic to improve their genetic algorithm. The list of items is sorted alternately: take the longest item, then the widest items according to the available width in the level, and so on.
- Divide Rule heuristic (DR): The longest item will form a level; the largest items will determine the shape of the layout, and the smallest items are favoured to fill gaps. Another possible improvement is to subdivide the list of items into two parts such that the first part contains a half number of items that are large (choosing alternately, longer item rather than wider item), and the second contains the other half of items taking it with their order of appearance in the original list. Thus, the large items are favored to be first. See Figure 7.

4. Experimental results

For our experiments, we firstly evaluate the quality of each fast greedy heuristic by applying the BLF2G guillotine placement heuristic to the engendered individuals, then evaluate the comportment of each fast greedy heuristic by combining it with the GA, which are developed in section 4.2., section 4.3. will describe our proposed method by using the fast greedy heuristics in the GA. Improvements are proposed and discussed in section 4.4.. for all our test we use the data set found in the literature, section 4.1.

4.1. Data sets found in the literature

To assess the performance of our new algorithm, we use the Msa datasets of Msabah and Baba-Ali [1], the C datasets of Hopper and Turton [6] and the N datasets of Burke et al. [4] (Table 1).

Table 1. Datasets found in the literature

	Name	# of Item	Plates dimension	Optimal height
Msabah and Baba-Ali [1]	Msa17(a, b, c)	17	200 x 200	200
	Msa35(a, b, c)	35	200 x 200	200
	Msa75(a, b, c)	75	200 x 200	200
	Msa150(a, b, c)	150	200 x 200	200
Hopper and Turton [6]	C1(1, 2, 3)	16 or 17	20 x 20	20
	C2(1, 2, 3)	25	40 x 15	15
	C3(1, 2, 3)	28 or 29	60 x 30	30
	C4(1, 2, 3)	49	60 x 60	60
	C5(1, 2, 3)	73	60 x 90	90
	C6(1, 2, 3)	97	80 x 120	120
	C7(1, 2, 3)	196 or 197	160 x 240	240
Burke et al. [4]	N1	10	40 x 40	40
	N2	20	30 x 50	50
	N3	30	30 x 50	50
	N4	40	80 x 80	80
	N5	50	100 x 100	100
	N6	60	50 x 100	100
	N7	70	80 x 100	100
	N8	80	100 x 80	80
	N9	100	50 x 150	150
	N10	200	70 x 150	150
	N11	300	70 x 150	150
	N12	500	100 x 300	300
	N13	3152	640 x 960	960

4.2. **Preliminary results**

Table 2 shows the height of the strip for each fast greedy heuristic; the best solutions are highlighted in bold and the optimal solution are highlighted by grey shadow; we can conclude that the DH policy and the DR policy are better and give often the best solution 44 times, in other hand the IH policy and the Harmonic policy are the worse with 0 best solution, the other policies gives weak results with 1, 2 and 4 best solution.

Table 2. Evaluation of the fast-greedy policies.

Name	DH policy	IH policy	DH-Reverse policy	IH-Reverse policy	Harmonic policy	DHOptW policy	DR policy
Msa17a	**240**	340	270	280	260	**240**	**240**
Msa17b	**245**	395	330	270	260	265	**245**
Msa17c	**263**	388	326	314	289	**263**	**263**
Msa35a	**220**	340	320	270	240	230	**220**
Msa35b	**225**	285	265	265	250	245	**225**
Msa35c	229	344	291	302	269	**223**	229
Msa75a	**214**	300	270	261	220	225	**214**
Msa75b	**210**	300	285	260	235	220	**210**
Msa75c	**210**	294	282	280	233	222	**210**
Msa150a	**205**	295	270	225	215	215	**205**
Msa150b	**205**	285	285	220	215	215	**205**
Msa150c	**218**	278	281	229	231	238	**218**
C11	**20**	29	33	28	31	24	**20**
C12	**25**	34	32	28	32	26	**25**
C13	25	34	**24**	26	28	26	25
C21	**17**	27	20	20	19	19	**17**
C22	**17**	23	20	19	18	21	**17**
C23	**16**	24	20	18	18	17	**16**
C31	**36**	49	41	**36**	39	**36**	**36**
C32	**36**	42	46	40	39	38	**36**
C33	**34**	59	46	44	36	35	**34**
C41	**72**	93	91	79	74	77	**72**
C42	**72**	103	101	96	81	73	**72**
C43	**63**	119	109	77	75	80	**63**
C51	**96**	120	127	112	105	99	**96**
C52	**102**	138	160	125	114	113	**102**
C53	**100**	154	115	113	113	105	**100**
C61	**130**	180	168	148	146	143	**130**
C62	**128**	205	179	159	142	156	**128**
C63	**135**	177	173	150	139	153	**135**
C71	**251**	321	304	275	275	274	**251**
C72	**250**	371	358	342	271	301	**250**
C73	**252**	366	345	296	275	294	**252**
N1	**40**	48	**40**	60	48	60	**40**
N2	**61**	87	69	65	63	63	**61**
N3	**53**	87	75	71	67	60	**53**
N4	**87**	147	131	148	104	106	**87**
N5	**109**	127	137	117	117	125	**109**
N6	**108**	120	135	123	114	110	**108**
N7	**118**	235	249	234	125	164	**118**
N8	**88**	172	134	107	93	106	**88**
N9	**158**	276	264	220	175	181	**158**
N10	**161**	283	216	212	164	157	**161**
N11	**156**	217	181	164	161	172	**156**
N12	**315**	448	446	366	326	366	**315**
N13	**973**	1159	1154	1004	1026	1022	**973**
Best result	**44**	0	2	1	0	4	**44**

Table 3. Evaluation of the fast-greedy policies combined with a genetic algorithm.

Name	BLF2G+GA	Generation	BLF2G+GA$_{DH}$	generation	BLF2G+GA$_{IH}$	generation	BLF2G+GA$_{DH-Reverse}$	generation	BLF2G+GA$_{IH-Reverse}$	generation	BLF2G+GA$_{Harmonic}$	generation	BLF2G+GA$_{DHOptW}$	generation	BLF2G+GA$_{DR}$	generation
Msa17a	200	2	200	0	200	44	200	24	200	22	200	45	200	49	200	27
Msa17b	200	41	200	91	200	129	200	78	200	19	200	2	200	128	200	1
Msa17c	200	5	200	3	200	2	200	4	200	3	200	3	200	6	200	3
Msa35a	200	241	200	403	200	71	200	10	200	194	200	208	200	92	200	8
Msa35b	210	60	210	177	210	119	210	28	210	215	210	65	210	13	210	3
Msa35c	218	3	218	4	213	164	215	361	218	32	218	5	218	5	215	541
Msa75a	207	691	205	811	210	22	207	495	210	19	207	829	207	740	205	1315
Msa75b	205	3	205	1	205	4	205	3	205	3	205	5	205	4	205	3
Msa75c	210	362	210	0	210	215	210	458	210	995	210	1231	212	927	210	0
Msa150a	205	0	205	0	205	0	205	0	205	0	205	0	205	0	205	0
Msa150b	205	0	205	0	205	2	205	2	200	1373	205	2	205	2	205	0
Msa150c	219	589	211	45	219	1110	219	193	216	539	218	502	217	466	210	25
C11	20	99	20	0	20	134	20	153	20	0	20	0	20	114	20	0
C12	22	11	22	1	22	26	22	14	22	0	22	3	22	30	22	19
C13	21	3	20	1	20	156	20	216	21	3	21	3	21	1	20	30
C21	16	42	16	1	16	8	16	4	16	26	16	20	16	5	16	0
C22	16	32	16	9	16	17	16	26	16	0	16	29	16	8	16	9
C23	15	70	15	15	15	43	15	7	15	107	15	29	15	23	15	12
C31	31	48	31	67	31	16	31	39	31	92	31	92	31	15	31	92
C32	33	4	33	5	33	17	33	184	33	47	33	94	33	36	33	107
C33	32	105	30	42	32	277	32	248	34	5	32	421	34	6	32	2
C41	65	278	65	27	65	123	65	149	65	69	65	48	65	215	65	6
C42	66	37	66	66	66	280	66	356	66	393	66	219	66	106	66	76
C43	64	359	62	6	64	24	64	60	64	12	64	11	64	132	63	0
C51	96	11	94	13	95	1122	95	823	95	1212	95	16	95	854	94	26
C52	97	446	97	1148	98	617	97	945	98	397	97	641	98	423	96	432
C53	95	529	95	6	95	896	95	164	95	816	95	533	95	539	94	477
C61	128	124	126	8	128	164	128	24	128	39	128	182	128	65	126	5
C62	131	55	126	0	131	75	130	29	131	47	130	70	131	19	126	0
C63	128	40	126	5	130	365	129	16	129	102	129	533	131	132	126	5
C71	257	917	251	0	258	1529	258	2001	260	3224	259	2926	257	1540	250	7
C72	261	496	249	1	259	2605	259	2637	261	3360	261	1802	260	3074	247	524
C73	256	2940	250	234	258	296	258	907	257	1046	257	802	256	3758	251	6
N1	40	0	40	0	40	0	40	0	40	0	40	0	40	0	40	0
N2	50	12	50	16	50	8	50	67	50	15	50	81	50	16	50	5
N3	53	13	53	0	53	46	53	34	53	43	53	33	53	12	53	0
N4	91	592	87	0	87	738	87	755	91	441	87	559	91	228	87	0
N5	106	77	106	22	106	557	106	622	106	340	106	278	106	437	105	776
N6	103	783	103	262	104	94	103	310	104	36	104	129	104	140	103	534
N7	116	18	116	4	116	0	116	7	116	5	116	16	116	7	116	5
N8	85	147	85	8	85	59	85	691	85	95	85	1124	85	165	84	91
N9	154	486	153	43	154	437	154	163	155	1286	153	1775	154	61	153	1030
N10	153	2742	152	2557	153	1818	153	509	153	941	153	757	153	738	153	1911
N11	155	824	153	61	155	741	155	990	155	3758	154	452	155	700	153	318
N12	319	104	310	232	319	287	318	493	319	292	315	1326	316	188	309	180
N13	1039	33	973	0	1052	31	1039	63	1004	0	1026	0	1022	0	973	0
Best Result	22		36		24		24		22		23		20		40	
Average		315		139		337		334		471		389		353		187

For more accurate we will see the comportment of these policies in the GA, as described above, by introducing them to the first population one by one separately, using the Visual C++ 6.0 programming language and all our experiments were run on a Windows computer with Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz and 15.9 GB RAM. We ran the GA Policy on each instance 10 times and keep the best fitness and mention the generation where this best fitness is found for the first time.

Table 3 shows the impact of the injection of the greedy policies to genetic algorithm. The authors find that the injection of the DR policy to genetic algorithm

(BLF2G+GA_{DR}) gives the best result, 40 times, followed by DH policy (BLF2G+GA_{DH}) 36 times. This mean that our new sorting Divide Rule policy outperform the famous Decreasing Height policy, which means that the DR policy managed successfully that characterized items appear first in the laying out process. In other hand the DH policy lead to the best solutions in an average of 139 generations, and the DR policy in an average of 187 generations, where the other methods need more time to give less satisfactory results in about 3 more times.

Table 4. Computational results on the datasets Msa, C and N

Name	BLF2G+GA	BLF2G+GA_{imprv}	BLF2G+GA_{imprv+}
Msa17a	**200**	**200**	**200**
Msa17b	**200**	**200**	**200**
Msa17c	**200**	**200**	**200**
Msa35a	220	215	**200**
Msa35b	215	210	210
Msa35c	219	213	215
Msa75a	215	205	210
Msa75b	210	205	205
Msa75c	218	210	210
Msa150a	205	205	205
Msa150b	205	205	205
Msa150c	219	212	214
C11	21	**20**	**20**
C12	22	22	22
C13	21	**20**	21
C21	16	16	16
C22	16	16	16
C23	16	**15**	**15**
C31	32	32	31
C32	34	33	33
C33	34	**30**	33
C41	67	64	65
C42	68	66	67
C43	64	62	63
C51	97	94	96
C52	102	97	97
C53	98	94	94
C61	133	126	128
C62	135	127	127
C63	133	126	127
C71	263	250	250
C72	266	248	248
C73	267	248	249
N1	**40**	**40**	**40**
N2	**50**	**50**	**50**
N3	55	53	53
N4	93	87	87
N5	106	106	107
N6	106	103	104
N7	116	116	116
N8	90	85	85
N9	158	153	153
N10	160	153	154
N11	158	153	154
N12	322	308	311
N13	1039	Out of service	1159

4.3. **Results**

In this section we propose to introduce all the sorting policies defined above to the initial population, in order to enrich the first generation by more qualified individuals. We generate one individual of every sorting policy, and we put all of them randomly in the first generation. We will compare the BLF2G guillotine placement heuristic combined with the new version of the GA, improved by this sorting policies, which we call BLF2G+GA_{imprv+}, to BLF2G+GA and BLF2G+GA_{imprv} proposed by Msabah and Baba-Ali [1].

Table 4 presents the height of the strip by applying our BLF2G+GA_{imprv+} algorithm with regard to BLF2G+GA and BLF2G+GA_{imprv}. The best results are highlighted in bold, the black shadows mean that the result has degraded with regard to the previous method, and when the optimum has reached the fitness are highlighted by grey shade.

The results in Table 4 show clearly that our new method, BLF2G+GA_{imprv+}, outperforms the BLF2G+GA method in all cases except for N5 and N13, i.e., guiding the evolutionary process by greedy heuristics improves the result in most cases. Compared to the BLF2G+GA_{imprv} method, our new method gives satisfactory results in a few cases, but it lost in most of the cases.

We can conclude that the evolutionary process was disturbed by the greedy policies. This means that the GA exploits the search space around these greedy heuristics (local optimum). To remedy that, we will present in the next section our improvements.

4.4. **Improvements**

Injecting sorted individuals into the population may improve the results but injecting additional sorted lists without a control may cause the search to converge to a local optimal solution, which disturbs the evolutionary process.

We propose a novel improvement to the genetic algorithm. We introduce a notion of the stability of the evolutionary process, such that, if there is no improvement after some number of iterations, we have obtained a local optimum solution. After stability is detected, we produce an individual of a randomly chosen sorting policy and we inject it randomly in the population. The chosen sorting policy is forbidden in the following injections, till all sorting policies are used. Then we continue the evolutionary process, while controlling the stability of the evolutionary process, until the end of the treatment.

```
Begin
Generate initial population
Nbr of generation :=0;
Nbr of stability := 0;
While Nbr of generation < maximum generation && optimal not reached
do
          Selection();
          Crossing();
          Mutation();
          Elitism();
          ++ Nbr of generation;
          If fitness = previous fitness
                    Then ++ Nbr of stability;
                    Else Nbr of stability := 0;
          End if
          If Nbr of stability = threshold
                    Then
                    Nbr of stability := 0;
                    Inject a sorted list of pieces to the population;
          End if
End while
End
```

- The controlled stability genetic algorithm (csGA).

This algorithm benefits from the order-based heuristics cited above and guides the genetic algorithm by introducing order-based heuristics to the population in turns, when stability is detected, in the evolutionary process. The factor of stability is defined as 5 times the number of items. A theoretical optimal solution can be reached when the square of the band, (ie. plate width * plate height), equal to the sum of the square of all pieces. Shown below is the genetic algorithm with controlled stability:

- The improved results.

For our experiments, we compare the new Controlled Stability GA "csGA" method combined with the BLF2G guillotine placement heuristic to BLF2G+GA and BLF2G+GA_{imprv} (Msabah et Baba-Ali, [1]), which use the guillotine constraint, and to the Fast layer-based heuristic (FH) algorithm (Leung et Zhang, [7]), which it was used without the guillotine constraint in the original work, with data sets found in the literature. We ran the csGA on each instance 10 times and keep the best-found solution.

Table 5 presents the height of the strip by applying our BLF2G+csGA method with regard to the FH and BLF2G+GA_{imprv} methods; the FH results are given by Leung et Zhang. N denote the number of items, W denote de width of the Bin and LB express the Lower Bound, i.e. the theoretical optimal solution. We mention the execution time expressed per second for our BLF2G+csGA method. We find that the GA process take benefit from some injected individuals and improve its quality.

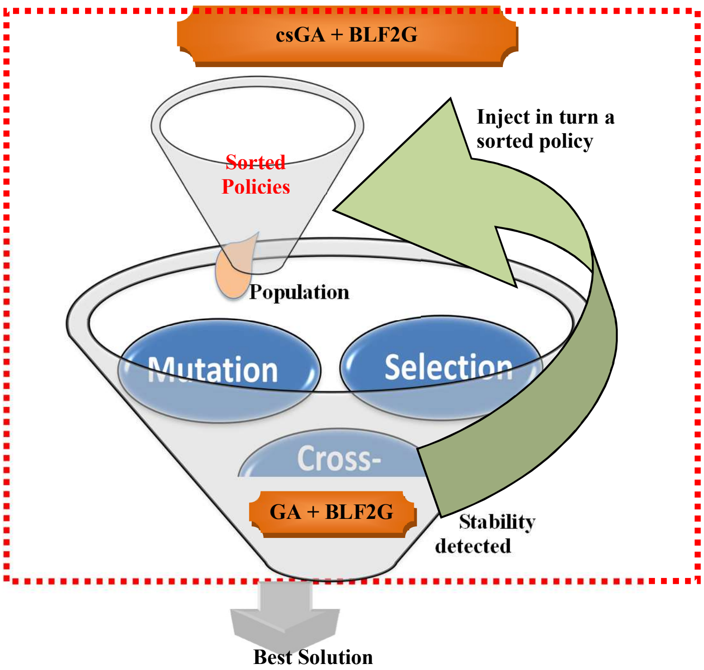

Fig. 8. The Controlled Stability Genetic Algorithm, csGA, process.

Table 5. Computational results on the datasets Msa, C and N

Name	Instance N	W	LB	FH	BLF2G+GA$_{imprv}$	BLF2G+csGA	Time (s)
Msa17a	17	200	200	210	**200**	**200**	**0**
Msa17b	17	200	200	220	**200**	**200**	**9**
Msa17c	17	200	200	**200**	**200**	**200**	**9**
Msa35a	35	200	200	210	215	**200**	**25**
Msa35b	35	200	200	**200**	210	210	**60**
Msa35c	35	200	200	**211**	213	213	**64**
Msa75a	75	200	200	**203**	205	205	**362**
Msa75b	75	200	200	**200**	205	205	**356**
Msa75c	75	200	200	**205**	210	208	**361**
Msa150a	150	200	200	**200**	205	205	**2505**
Msa150b	150	200	200	**200**	205	**200**	**2513**
Msa150c	150	200	200	**204**	212	210	**2515**
C11	16	20	20	**20**	**20**	**20**	**4**
C12	17	20	20	**20**	22	22	**18**
C13	16	20	20	21	**20**	**20**	**16**
C21	25	40	15	**16**	**16**	**16**	**35**
C22	25	40	15	**15**	16	16	**37**
C23	25	40	15	**15**	**15**	**15**	**5**
C31	28	60	30	**31**	32	**31**	**44**
C32	29	60	30	**31**	33	33	**47**
C33	28	60	30	32	**30**	**30**	**43**
C41	49	60	60	**61**	64	63	**148**
C42	49	60	60	**61**	66	63	**256**
C43	49	60	60	**61**	62	62	**247**
C51	73	60	90	**91**	94	94	**503**
C52	73	60	90	**90**	97	95	**763**
C53	73	60	90	**91**	94	94	**702**
C61	97	80	120	**121**	126	125	**810**
C62	97	80	120	**121**	127	126	**564**
C63	97	80	120	**121**	126	126	**802**
C71	196	160	240	**241**	250	249	**4572**
C72	197	160	240	**241**	248	246	**4476**
C73	196	160	240	**241**	248	248	**4992**
N1	10	40	40	**40**	**40**	**40**	**0**
N2	20	30	50	52	**50**	**50**	**1**
N3	30	30	50	**51**	53	53	**45**
N4	40	80	80	**83**	87	87	**93**
N5	50	100	100	**102**	106	105	**144**
N6	60	50	100	**101**	103	103	**227**
N7	70	80	100	**102**	116	103	**357**
N8	80	100	80	**81**	85	84	**514**
N9	100	50	150	**151**	153	152	**722**
N10	200	70	150	**151**	153	152	**5510**
N11	300	70	150	**151**	153	153	**17438**
N12	500	100	300	**301**	308	308	**83525**
N13	3152	640	960	**960**	Out of service	973	**604800**
			Optimum reached:	12	9	11	
			Best results:	40	10	13	
Best results BLF2G+GA$_{imprv}$vsBLF2G+csGA				/	28	46 (all time)	

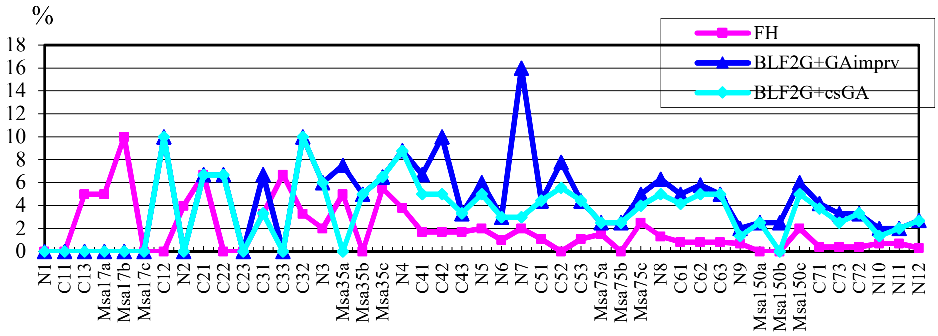

Fig. 9. Comparison of the BLF2G+csGA to FH and BLF2G+GA_{imprv} algorithms

Test sets sorted according to the number of items

With this improvement, our BLF2G+csGA method maintains its superiority over the BLF2G+GA_{imprv} method in all cases (which work in the same conditions); the results are 46 vs 28 in favor of the csGA algorithm, which mean an improvement of 139%. Despite our method using the guillotine constraint, we reach the optimum 11 times, especially in Msa35, C13, C33 and N2, where the FH algorithm fails to reach the optimum despite its freedom from the guillotine constraint. In other cases, the FH algorithm outperforms our method.

To show the comparisons more clearly, we give the percentage of loss in Figure 9, As seen in the graph, our method gives the best results for datasets of small size, and it is still competitive for the rest.

5. CONCLUSION

This paper demonstrated the efficiency of our contribution to the rectangular cutting-stock problem. The authors used the new BLF2G guillotine placement heuristic combined with a genetic algorithm guided intelligently by sorted lists of items. Introducing several ordered lists to the evolutionary process might disturb the quality of the treatment to a local optimal solution. To remedy this issue, we propose to guide the genetic algorithm intelligently. In addition to the famous DH and DHOptW policies, we developed new ordered policies. We inject these sorted lists into the population when we observe that there is stagnation in the evolutionary process, i.e. we are in a local optimal solution. This will help diversify the search in another area of the solution space.

The comparisons between our csGA combined with the BLF2G heuristic and other methods are very encouraging. Our method repeatedly reaches the optimum, especially for Msa35a and Msa150b with 35 and 150 items. In comparison to the FH algorithm, our method gives better results in a few cases, despite FH's freedom from the guillotine constraint, but it still competitive for most other cases.

The BLF2G heuristic depends completely on the GA to find the order of items that yields the optimal solution. Future work, can intend to enhance the GA in the genetic phase without altering the evolutionary process, by proposing new ordered based heuristics to give an intelligent order of items especially, after the satisfactory results of the DR heuristics to help the evolutionary process to find the best solution.

References

[1] Msabah, S. A., and A. R. Baba-Ali. 2011. A New Guillotine Placement Heuristic Combined with an Improved Genetic Algorithm for the Orthogonal Cutting-Stock Problem, *Proceedings of The IEEE International Conference on Industrial Engineering and Engineering Management, IEEM11, Singapore*, 482-486.

[2] Lodi A., S. Martello, and D. Vigo. 1999. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems, *INFORMS journal on computing* ***Vol 11***, 345–357.

[3] Ben Messaoud, S., C. Chu, and M. L. Espinouse. 2004. An approach to solve cutting stock sheets, *IEEE International Conference on Systems, Man and Cybernetics*, 5109–5113.

[4] Burke E. K., G. Kendall, and G. Whitwell. 2004. A New Placement Heuristic for the Orthogonal Stock Cutting Problem, *Operations Research* ***vol. 52 no. 4***, 655–671.

[5] Hopper E., and B. Turton. 1999. A genetic algorithm for a 2D industrial packing problem, *Computers and Industrial Engineering* ***vol. 37/1-2,*** 375–378.

[6] Hopper E., B. 2001. Turton. An empirical investigation of metaheuristic and heuristic algorithms for a 2D packing problem, *European Journal Operational Research* ***128***, 34–57.

[7] Leung S.C.H., and D. Zhang. 2011. A fast layer-based heuristic for non-guillotine strip packing, *presented at Expert Syst. Appl.* 13032-13042.

[8] Goldberg, D.E. *Genetic algorithms in search, optimization, and machine learning,* Reading, MA: Addison-Wesley, 1989.

[9] Michalewics, Z. 1996. *Genetic Algorithms + Data Structures = Evolution Programs,* Third, Revised and Extended Edition, Springer.

[10] Baker, B. S.; Coffman, E. G.; Rivest, R. L. (1980) Orthogonal packings in two dimensions, SIAM Journal of Computing 9, 4, pp. 846–855.

[11] Jakobs, S. (1996) On genetic algorithms for the packing of polygons, European Journal of Operational Research n° 88, pp. 165–181.

[12] Liu, D.; Teng, H. (1999) An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles, European Journal of Operational Research 112, pp. 413–420.

[13] Ramesh Babu, A.; Ramesh Babu N. (1999) Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms, International Journal of Production Research Vol. 37, n°7, pp. 1625–1643.

[14] Burke, E. K.; Kendall, G.; Whitwell, G. (2009) A Simulated Annealing Enhancement of the Best-Fit Heuristic for the Orthogonal Stock-Cutting Problem, INFORMS Journal on Computing Vol. 21, No. 3, pp. 505–516.

[15] Berkey, J. O.; Wang, P. Y. (1987) Two dimensional Finite bin packing algorithms, Journal of the Operational Research Society 38, pp. 423–429.