Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 167 (2020) 809–818

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Intelligence and Data Science (ICCIDS 2019)

A Binary Crow Search Algorithm for Solving Two-dimensional Bin Packing Problem with Fixed Orientation

Soukaina Laabadi[a]*, Mohamed Naimi[b], Hassan El Amri[a] and Boujemâa Achchab[b]

[a] *Laboratory of Mathematics and Applications, ENS- Hassan II University, Casablanca, Morocco*
[b]*Laboratory of Analysis, Modeling Systems and Decision Support, ENSA- Hassan I University, Berrechid, Morocco*

Abstract

This work aims to propose a binary version of a new metaheuristic in order to solve the two-dimensional bin packing problem (2D-BPP) which is a NP-hard optimization problem in a strong sense. Various metaheuristics are dedicated to solve 2D-BPP, however, the swarm intelligence algorithms are rarely frequent in this problem. A new algorithm based on the principles of swarm intelligence was recently introduced, called crow search algorithm (CSA), which has been successfully applied to continuous optimization problems. In this paper, the CSA is binarized by applying a sigmoid transformation so as to adapt it for solving the 2D-BPP. An evaluation of the performance of the proposed binary CSA is carried out by standard instances of 2D-BPP, and the experimental results are compared with two state-of-the-art algorithms that are particle swarm optimization algorithm and the well-known heuristic Finite First Fit. The results obtained by the proposed algorithm prove its effectiveness in terms of solution quality and computational time. Until recently, no study exists that deals with CSA algorithm for solving the packing problems in general and 2D-BPP in particular.

Keywords: Two-dimensional bin packing problem; Binary crow search algorithm; Binarization process; Sigmoid function; Repair operator;

* Corresponding author. Tel.: +212-645-700-469
E-mail address: soukainalaabadi@gmail.com

1877-0509
10.1016/j.procs.2020.03.420

1. Introduction

Given a set of n rectangular-shaped items, each characterized by a width w_i and a height h_i, where $i \in I = \{1, \ldots, n\}$, as well as an unlimited number of identical two-dimensional containers, called bins, having width W and height H. The two-dimensional bin packing problem (2D-BPP) consists of packing all items in as few bins as possible so that no two items overlap and without violating the capacity ($W \times H$) of each bin. In this work, it is assumed that the items have a fixed orientation, *i.e.* they cannot be rotated. It is also assumed, without loss of generality, that the width and the height of items are non-negative integers satisfying, respectively, $w_i \leq W$ and $h_i \leq H$.

The 2D-BPP is a NP-hard combinatorial optimization problem in the strong sense [1], because it is a generalization of the well-known one-dimensional bin packing problem (1D-BPP), in which only one dimension is considered (width or height). Several variants can be found in the 2D-BPP literature: The two-dimensional strip packing [1] that considers a vertical strip of fixed width. The goal is to minimize the height needed to pack a given set of rectangular-shaped items. The two-dimensional vector packing problem [2] that aims to pack all items in the minimum number of unit-bins (in which both width and height are equal to 1). Other variants of the 2D-BPP arise when additional constraints are imposed. Generally, the constraints reflect conditions of the arrangement of items into the bins such as requiring guillotine cuts [3], allowing the items to be rotated by 90° [4], or forbidding conflicting items to be packed in the same bin [5]. In industrial settings, further constraints may be imposed, such the case of glass-cutting industry [6], or how to ship fragile items ([5], [7]).

A significant part in the literature of packing problems has been devoted to the 2D-BPP due to its practical interest. [1] have surveyed available lower bounds, exact algorithms and approximate methods for solving 2D-BPP up to 2002. Notice that lower bounds are important both in the implementation of exact methods and in the empirical evaluation of approximate solutions. Many methods [8] have been proposed for computing lower bounds. However, there are few articles dealing with exact methods. [9] have combined a new lower bound with an enumerative algorithm to find exact solutions. [10] have proposed two exact algorithms: The first algorithm is an improved version of the well-known branch and bound method, while the second algorithm is based on a new relaxation of the problem. In contrast, heuristic and metaheuristic approaches have attracted the interest of many researchers. [11] have developed level-oriented algorithms that extend the well-known heuristics of packing, namely First Fit (FF), Next Fit (NF) and Best Fit (BF), to 2D-BPP. Knowing that FF, NF and BF were originally proposed to strip packing problem that consists of packing items in a single bin with unlimited height. [12] have proposed a two-phase heuristic, called the knapsack packing (KP). In the first phase, the authors pack the items into levels by solving series of knapsack problems. In the second phase, they pack the obtained levels into bins. [13] has introduced a new heuristic based on the concept of maximal area to manage the free areas in the bins. The empty area is maximal if it is not contained in any other area in the bin. As far as metaheuristics are concerned, [14] have hybridized a novel placement procedure with a random key genetic algorithm (RKGA). In the RKGA, the chromosomes are represented as vectors of randomly generated real numbers in the interval [0,1]. Besides, [13] has combined its proposed maximal area based heuristic with ant colony optimization algorithm. [15] have proposed a hybrid genetic algorithm and a hybrid simulated annealing to solve 2D-BPP by making a comparison between them. Moreover, [16] have proposed a hybrid approach for solving two and three-dimensional bin packing problem. Their approach combines Greedy Randomized Adaptive Search Procedure (GRASP) with Variable Neighborhood Descent (VND), and uses the concept of maximal space in the constructive phase. Nevertheless, [12] have proposed a tabu search that uses different inner heuristics separately to explore the neighborhood. [17] have designed an algorithm based on ant colony system to solve 2D-BPP with possibility to rotate items by 90°. More recently, [18] have applied particle swarm optimization algorithm to solve 2D-BPP by considering it as a continuous-value optimization problem and not as the discrete-value optimization problem.

The structure of this paper is organized as following: Section 2 introduces and elaborates standard CSA. Section 3 presents the proposed binary version of CSA for solving 2D-BPP. Section 4 shows the experimental results of the proposed algorithm by comparing it with two other algorithms. Section 5 presents the main conclusions.

2. Introduction to crow search algorithm

CSA is a novel metaheuristic inspired by the intelligence of crows. It is based on the idea that crows store their excess foods in hiding places and retrieve them in case of need. CSA is shown to be very effective in solving continuous

optimization problems, such as pressure vessel design problem and gear train design problem [19]. It is assumed that there is a d-dimensional environment including N crows. The position of each crow i at time t in the search space having d dimensions, is presented by a vector $x_i(t) = (x_i^1(t), x_i^2(t), \dots, x_i^d(t))$, where $i \in \{1, \dots, N\}$ and each x_i represents a feasible solution of the optimization problem. In addition to positions, each crow i also has a memory $m_i(t)$ at time t, and can hence remember the best position in which it has stored its food. The feasibility of the crows' position is evaluated by the fitness function or the objective function of the optimization problem.

Based on thievery behavior that crows exhibit, the crows update their positions in the search space as following: Crow i randomly selects crow j from the flock of crows, and follows it to discover where the latter has stored its food. The new position of the crow i is obtained by the following equation:

$$x_i(t) = \begin{cases} x_i(t-1) + fl_i \times r_i \times (m_j(t-1) - x_i(t-1)) & if\ r_j \geq AP_j \\ random\ position & otherwise \end{cases} \quad (1)$$

Where fl_i is the flight length of the crow i, AP_j is the probability of awareness of the crow j. Besides, r_i and r_j are random numbers in the interval [0,1]. The awareness probability value controls the diversification and the intensification of CSA.

After checking the feasibility of the new crows' position, each crow updates its memory by the equation (2). If the new position is infeasible, the crow stays in its current position.

$$m_i(t) = \begin{cases} x_i(t) & if\ \ f(x_i(t)) \geq f(m_i(t-1)) \\ m_i(t-1) & otherwise \end{cases} \quad (2)$$

Where $f(.)$ is the objective function or the fitness function value.

The CSA is stopped when a maximum number of iterations is reached. In this step, the best memory in terms of fitness function (or the objective function), is reported as the solution of the optimization problem.

3. Binary crow search algorithm

As mentioned in the previous section, the CSA has a continuous nature. So, CSA cannot settle the binary problems straightforwardly such as 2D-BPP. For this reason, this paper proposes a binary version of crow search algorithm (BCSA) and adapts it to the context of 2D-BPP. The following subsections will elaborate the sequential mechanisms in the proposed algorithm. The main steps of BCSA are illustrated in the flowchart of Fig.1.

3.1. Initialization of crows swarm

At the initialization stage, N crows are randomly positioned in d-dimensional search space. The crow position (solution) is presented by a binary matrix of d columns and n rows, where n is the number of items to be packed into bins, and d corresponds to the number of available bins. The representation of each solution i at time t is defined as follows:

$$x^i(t) = \left\{ x_{jk}^i(t) \ /\ \ 1 \leq j \leq n \ \ and \ \ 1 \leq k \leq d \right\};\ \forall i \in \{1, \dots, N\} \quad (3)$$

Where
$$x_{jk}^i(t) = \begin{cases} 1 & if\ the\ item\ j\ is\ packed\ in\ the\ bin\ k \\ 0 & otherwise \end{cases} ;\ \forall i \in \{1, \dots, N\} \quad (4)$$

Notice that the memory of each crow is initialized by its initial position, since it had as yet no experience in this stage.

3.2. Checking the feasibility of solutions

Since the position of crows is randomly generated, some infeasible solutions may occur in search space. A solution is considered as infeasible when it violates the capacity constraints of used bins. Therefore, for dealing with infeasible solutions, a problem specific-knowledge repair operator is developed. This repair operator is applied only to overloaded bins. It consists firstly of removing items from overloaded bins in turn so as to respect their loading capacities. Then, it inserts the removed items one by one in the former well-filled bins that can accommodate them, by using the Bottom-Left policy [11] that consists of placing items in the lowest and the left-most position. If there is no bin in which the removed items can be packed, a new bin is created.

3.3. Binarization process

The equation (1) is used to update the crow positions, which generates real number positions. Thus, in order to convert real valued solutions into binary ones, a sigmoid function (see [20]) is applied:

$$sig(x) = \frac{1}{1-e^{-x}} \; ; \; \forall x \in \square \tag{5}$$

$$x_{binary} = \begin{cases} 1 & if \;\; sig(x) \geq rand() \\ 0 & otherwise \end{cases} \tag{6}$$

Where $rand()$ is a random number selected from a uniform distribution in [0,1]. Therefore, the resulting change in crow's position is defined by the following rules:

I. If $r_j \geq AP_j$

i. Calculate the following funtion :

$$g^i(t-1) = fl_i \times r_i \times (m^j(t-1) - x^i(t-1)) \tag{7}$$

ii. Calculate the sigmoid function of equation (7) :

$$sig(g^i(t-1)) = \frac{1}{1-e^{-g^i(t-1)}} \tag{8}$$

iii. Then, apply equation (6) to discretize x^i at time t.

II. Else

For each crow i, its corresponding position is randomly updated according to following rule:

$$\forall 1 \leq k \leq d \;\; and \;\; 1 \leq j \leq n \; ; \quad x^i_{jk}(t) = \begin{cases} 1 & if \;\; rand() \geq 0.5 \\ 0 & otherwise \end{cases} \tag{9}$$

Then, the feasibility of the generated solutions is evaluated thanks to a fitness function (see the next section). If it is necessary, the new proposed repair operator is applied for restoring the feasibility of solutions.

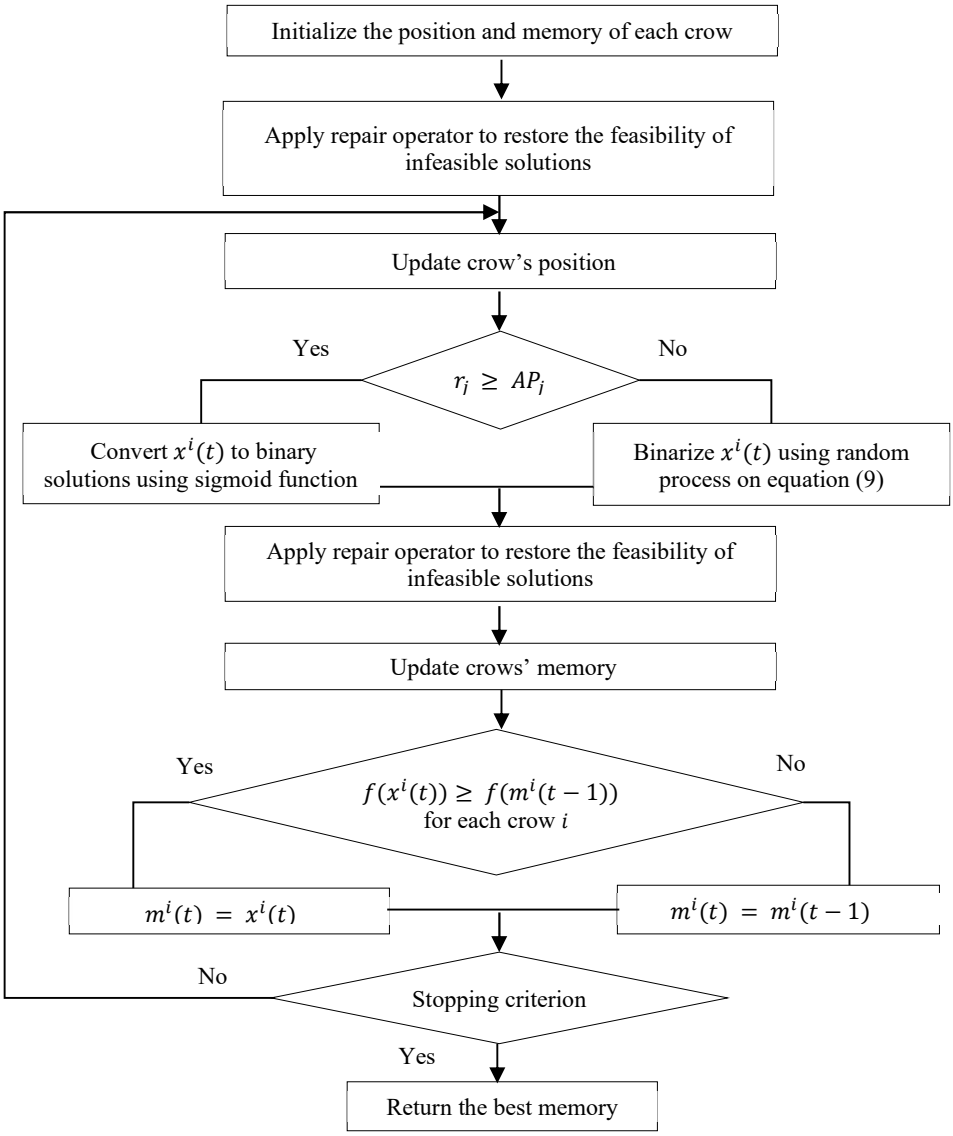

Fig. 1. Flowchart of BCSA

3.4. Evaluation of solutions

In the most case, the objective function of the bin packing problems differs from the fitness function. In fact, minimizing the number of bins used to pack all items is correct from a mathematical point of view but it is unusable in practice (see [21]). Thus, to evaluate and assess the quality of solutions, the fitness function introduced by [21] is used by adapting it to the context of 2D-BPP:

$$f = \frac{\sum_{u=1}^{usedBins} (fill_u / C)^z}{usedBins} \tag{10}$$

Where $usedBins$ is the number of bins used to pack the total amount of surface area covered by the items placed in bin u. C is the capacity of each bin in terms of surface, and z is a constant (> 1) which indicates the concentration

in the well-filled bin compared to the less-filled bin. The experimental results carried out by the authors have shown that $z = 2$ yields good results. This fitness function is to be maximized. It is based on the fact that fulfilling bins as maximum as possible with available items may reduce the number of used bins.

3.5. Updating of crows memory

Once the quality of solutions is evaluated by the equation (10), the memory of each crow at time t is updated with equation (2). If the new position of a crow is better than the most recently memorized position, the crow updates its memory with this new position.

3.6. Termination criterion

The previous steps are repeated until a maximum iteration time is met. As aforementioned, an individual crow might have best hiding place (memory) at every iteration. Hence, at the maximum iteration, the best memory in swarm of crows is reported as a global optimum solution.

4. Experimental study

4.1. Experimental design

In order to verify the effectiveness of the proposed BCSA, the latter is tested on 2D-BPP benchmarks available at http://or.dei.unibo.it/library/two-dimensional-bin-packing-problem. In fact, the 2D-BPP benchmark instances are categorized in 10 classes. The first six classes are introduced by [11] and the last four classes are proposed by [9] including more realistic instances. Each class has a fixed and identical dimension of bins which are 10×10, 30×30, 40×40, 100×100 and 300×300, while the height and the width of each item, are randomly generated following a uniform distribution in a specific interval. Each class is divided into five subclasses according to the number of items to be packed ($n = 20, 40, 60, 80, 100$). Furthermore, each subclass contains ten problem instances. So, the benchmark instances contain globally 500 problem instances. For simulation experiments, one problem instance is randomly selected from each subclass by considering it as a representative individual, giving a total of 50 test instances instead of 500 test instances. Furthermore, the proposed BCSA is compared against two algorithms, namely; the binary particle swarm optimization algorithm (BPSO) of [20], which has undergone some modifications to adapt it to the context of the 2D-BPP, and the well-known Finite First Fit heuristic (FFF) of [11]. All these previous algorithms are implemented in JAVA programming language using a PC with Intel® Core™ i5, 2.5 GHz and 4.0 Go of RAM, running on 64-bits Windows 7 operating system. The control parameters in BCSA and BPSO are predefined for all runs (see Table 1).

Table 1. Parameters setting.

Algorithms	Parameters	Value
BCSA	Flock size	50
	Flight size	2
	Awareness probability	0,1
	Maximum iteration	100
BPSO	Particle swarm size	50
	Maximum / minimum inertia weight	w_{min} = 0,4; w_{max}= 0,9
	Acceleration coefficients	$c_1 = c_2$ = 1,5
	Velocity	V_{min} = -2; V_{max} =2
	Maximum iteration	100

4.2. Experimental results

Due to stochastic nature of BCSA and BPSO, 20 independent runs have been executed. The comparison is investigated based on two performance measures: average fitness value (AVG), and running time (CPU) expressed in seconds. AVG indicates the average fitness value over 20 successive runs, while CPU reflects the average speed needed to find this latter. Table 2 and 3 summarize the comparison of BCSA against FFF and BPSO. In each table, the first and second columns indicate, respectively, the problem class and its respective bin dimensions. The next pair of columns report, respectively, the selected test instances and their corresponding number of items, while the last columns report the computational results of the proposed algorithm and its rival methods in terms of AVG and CPU.

The experimental results show that BCSA performs much better than both algorithms in terms of AVG in all instances (bold values indicate the best values among algorithms). In other words, BCSA exploits more effectively the used bins (*i.e* it places the items in a tight space), which allows reducing the number of bins to be used for packing all available items. As far as runtime is concerned, BCSA needs a few time to find best solutions in comparison of BPSO. However, FFF yields good results, but not better than those of BCSA in extremely rapid time. As can be observed from tables, the BCSA yields better results independently of the number of items and the bins capacity.

Table 2. Comparison of BCSA against two comparative methods by using Berkey and Yang instances

Problem class	$H \times W$	N° of instance	n	FFF		BPSO		BCSA	
				CPU	AVG	CPU	AVG	CPU	AVG
Class I	10 × 10	Inst8	20	0,0008	419,69	0,6185	389,04	0,0289	**554,94**
		Inst10	40	0,0000	3442,92	3,2768	3117,98	0,2076	**4788,42**
		Inst5	60	0,0000	13133,92	8,1276	12141,97	0,8858	**16948,96**
		Inst4	80	0,0008	36814,28	16,9325	33332,82	2,3750	**47425,40**
		Inst2	100	0,0008	73044,58	32,4253	67318,06	6,4151	**95381,72**
Class II	30 × 30	Inst1	20	0,0000	10,37	0,5795	9,48	0,0253	**17,99**
		Inst3	40	0,0000	93,92	2,3564	87,78	0,2192	**175,12**
		Inst6	60	0,0008	201,26	8,1266	186,12	1,6102	**274,21**
		Inst7	80	0,0008	684,73	20,0036	634,15	2,9041	**973,27**
		Inst2	100	0,0016	1119,59	32,7567	1044,55	6,3869	**1709,18**
Class III	40 × 40	Inst4	20	0,0000	170,32	0,8095	151,36	0,0262	**268,33**
		Inst6	40	0,0004	1671,88	2,4936	1540,05	0,2132	**2214,68**
		Inst3	60	0,0005	7721,35	7,4652	6956,18	0,9835	**9551,37**
		Inst9	80	0,0016	21299,40	21,6988	19574,45	2,8330	**33326,25**
		Inst8	100	0,0031	36804,78	33,6686	34175,16	6,3895	**50155,57**
Class IV	100 × 100	Inst5	20	0,0000	6,09	0,5546	5,80	0,0336	**12,22**
		Inst8	40	0,0000	99,43	2,4059	92,23	2,4415	**200,20**
		Inst3	60	0,0008	228,21	7,5745	212,02	1,4555	**347,19**
		Inst10	80	0,0016	529,44	16,6386	497,30	2,6485	**1029,12**
		Inst4	100	0,0031	916,19	33,8111	855,79	6,0642	**1538,55**
Class V	100 × 100	Inst10	20	0,0008	513,44	0,4360	494,87	0,0310	**551,35**
		Inst2	40	0,0001	2662,12	2,2714	2435,58	0,2069	**3141,19**
		Inst8	60	0,0002	12323,78	6,8261	11520,90	0,8137	**18568,14**
		Inst1	80	0,0005	23701,70	16,7183	21225,96	2,6943	**35776,63**

		Inst7	100	0,0006	44371,01	33,3851	40885,44	5,9294	**55431,63**
		Inst5	20	0,0000	4,70	0,4415	4,49	0,0321	**9,63**
		Inst6	40	2,0000	37,84	2,3004	35,74	0,2427	**70,63**
Class VI	300 × 300	Inst3	60	0,0000	175,10	6,5669	163,43	0,8489	**216,42**
		Inst4	80	0,0008	406,93	20,4363	381,47	2,8452	**578,74**
		Inst2	100	0,0000	833,20	31,5641	782,07	6,1759	**1160,02**

Table 3. Comparison of BCSA against two comparative methods by using Martello and Vigo instances

Problem class	$H \times W$	N° of instance	n	FFF		BPSO		BCSA	
				CPU	AVG	CPU	AVG	CPU	AVG
		Inst5	20	0,0000	359,66	0,4416	262,12	0,0351	**513,20**
		Inst10	40	0,0008	3325,23	2,7616	2515,77	0,1455	**3547,29**
Class VII	100 × 100	Inst2	60	0,0000	7764,81	8,2844	5993,76	0,9960	**9189,37**
		Inst9	80	0,0008	25174,89	20,7807	19563,61	3,0081	**31351,27**
		Inst3	100	0,0003	36327,73	43,4908	26891,87	8,1283	**46705,27**
		Inst7	20	0,0000	201,20	0,5484	194,47	0,0240	**452,12**
		Inst2	40	0,0000	3625,57	2,7184	3534,28	0,2202	**4779,44**
Class VIII	100 × 100	Inst9	60	0,0000	9303,43	7,3168	9149,17	0,8273	**15102,90**
		Inst4	80	0,0008	19884,02	19,9404	19359,51	2,5199	**30795,66**
		Inst10	100	0,0008	59385,17	33,9708	58345,50	5,9369	**87019,54**
		Inst1	20	0,0000	1164,18	0,4480	1164,18	0,0234	**1979,89**
		Inst8	40	0,0000	7591,02	2,2238	7455,61	0,2186	**13881,30**
Class IX	100 × 100	Inst3	60	0,0000	27988,11	6,7419	26690,61	0,8955	**49495,61**
		Inst9	80	0,0000	55743,95	20,3263	53357,95	2,5225	**84037,03**
		Inst4	100	0,0008	119126,66	32,6363	113640,01	6,5365	**160066,32**
		Inst10	20	0,0000	72,62	0,4505	68,61	0,0211	**150,58816**
		Inst5	40	0,0000	935,21	2,3374	884,41	0,2318	**1048,6964**
Class X	100 × 100	Inst7	60	0,0000	4338,71	8,0474	4121,32	0,9509	**7150,3205**
		Inst1	80	0,0000	9885,01	19,8340	9155,39	2,5715	**16879,532**
		Inst8	100	0,0008	8225,35	32,7937	7664,58	6,0092	**11745,764**

Figs. 2-4 depict a comparison test of BCSA against BPSO and existing lower bounds (LB*) in the literature (see http://or.dei.unibo.it/general-files/best-known-solution-and-lower-bound-each-instance). The comparison is performed based on the average number of bins used in each instance (plot on the left) and the time required for each instance to get the best solution (plot on the right). The experiments are tested on class IX of [9]. The graph on left in each figure is plotted with the average used bins as the horizontal axis and the ten test instances of each sub-class as the vertical axis, while the graph on the right is plotted with the average running time as the horizontal axis and the ten test instances as the vertical axis. All the test problems are evaluated by 20 independent runs, and the average results are considered for graphic illustration. From Figs. 2-4, it can be clearly seen that BCSA yields best results so near from lower bounds and sometimes better than them. It is also seen that the proposed algorithm needs few bins to pack all items in comparison with BPSO, and takes less time than it, regarding all test instances.

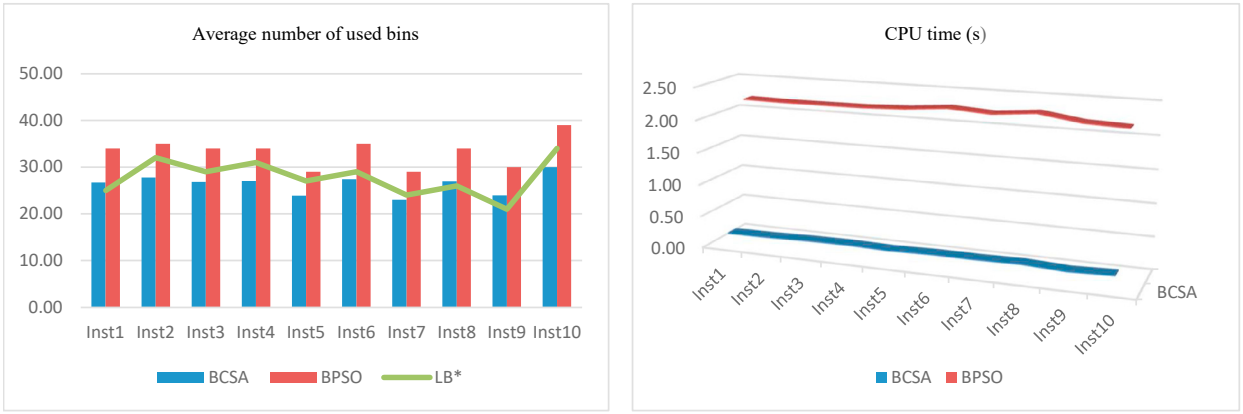

Fig. 2.Comparison of BCSA (n = 40)

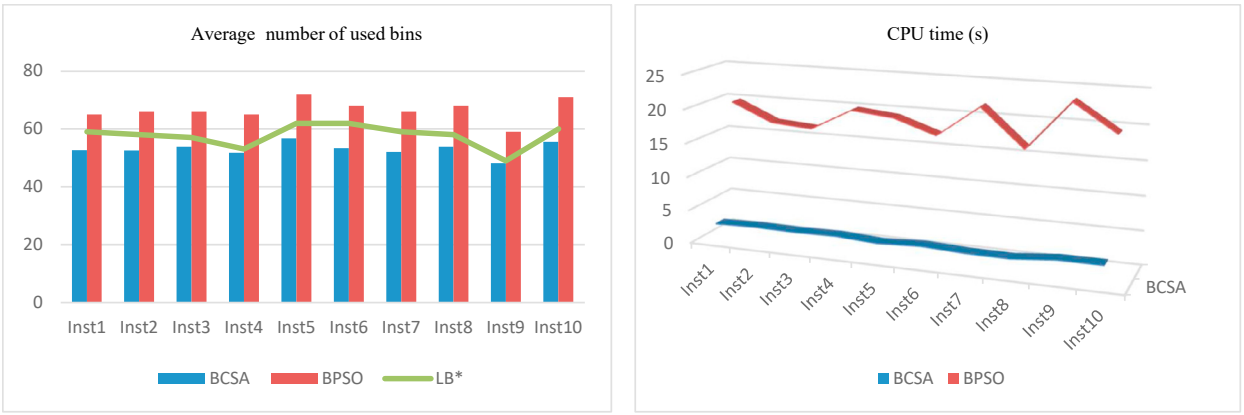

Fig. 3. Comparison of BCSA (n = 80)

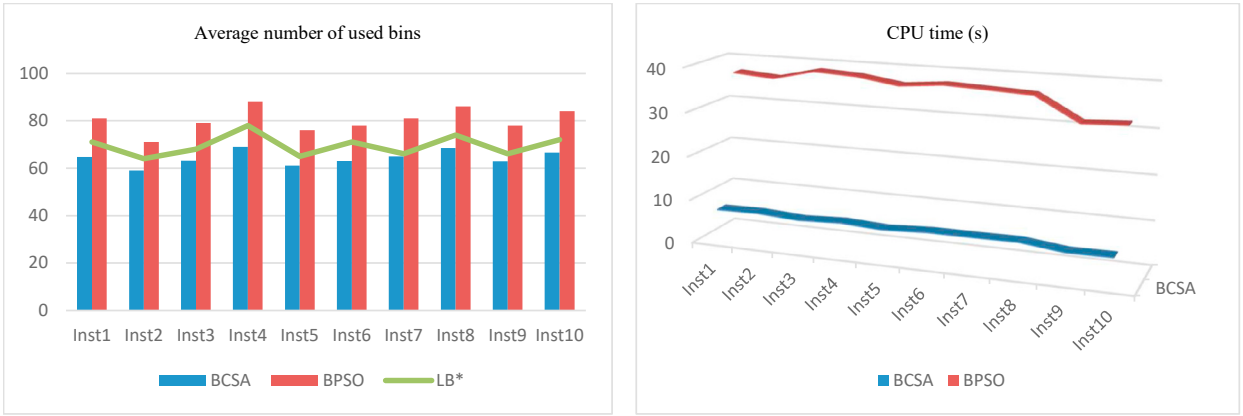

Fig. 4. Comparison of BCSA (n = 100)

5. Conclusions

Throughout this article a binary crow search algorithm is proposed for solving a NP-hard problem, that is the two-dimensional bin packing problem. In discrete process, a sigmoid function is used for mapping real-valued solutions into binary ones. In addition, a problem specific knowledge repair operator is introduced in order to increase the diversity of solutions. The performance of the proposed algorithm is then examined on two data sets of benchmark instances with different structures. In comparison with a heuristic algorithm and a bio-inspired algorithm, the proposed algorithm BCSA is faster than the bio-inspired algorithm and achieves better numerical performance against both comparative methods. In future research, the BCSA would be applied and validated in other important binary optimization problems such as the multi-dimensional knapsack problem or other variants of bin packing problem, like bin packing with fragile objects or with conflict constraints. Moreover, other binarization techniques can be incorporated in the crow search algorithm for working in binary space.

References

[1] Lodi, Andrea, Silvano Martello, and Daniele Vigo. (2002) "Recent advances on two-dimensional bin packing problems." *Discrete Applied Mathematics* **123(1-3)**: 379-396.

[2] Spieksma, Frits CR. (1994) "A branch-and-bound algorithm for the two-dimensional vector packing problem." *Computers & operations research* **21(1)**: 19-25.

[3] Charalambous, Christoforos, and Krzysztof Fleszar. (2011) "A constructive bin-oriented heuristic for the two-dimensional bin packing problem with guillotine cuts." *Computers & Operations Research*: **38(10)**, 1443-1451.

[4] Clautiaux, François, Antoine Jouglet, and Joseph El Hayek. (2007) "A new lower bound for the non-oriented two-dimensional bin-packing problem." *Operations Research Letters* **35(3)**: 365-373.

[5] Khanafer, Ali. (2010) "*Algorithmes pour des problèmes de bin-packing mono et multi-objectif* " *Ph. D. thesis*, Lilles.

[6] Puchinger, Jakob, Günther R. Raidl, and Gabriele Koller. (2004) "Solving a real-world glass cutting problem", In *European Conference on Evolutionary Computation in Combinatorial Optimization*, Springer, Berlin, Heidelberg.

[7] Laabadi, Soukaina. (2016) "A new algorithm for the Bin-packing problem with fragile objects", In *2016 3rd International Conference on Logistics Operations Management*, IEEE, 1-7.

[8] Clautiaux, François. (2005) "*Bornes inférieures et méthodes exactes pour le problème de bin packing en deux dimensions avec orientation fixe*". *Ph. D. thesis*, Université de Technologie de Compiègne.

[9] Martello, Silvano, and Daniele Vigo. (1998) "Exact solution of the two-dimensional finite bin packing problem." *Management science* **44(3)**: 388-399.

[10] Clautiaux, François, Jacques Carlier, and Aziz Moukrim. (2007) "A new exact method for the two-dimensional orthogonal packing problem." *European Journal of Operational Research* **183(3)**: 1196-1211.

[11] Berkey, Judith O., and Pearl Y. Wang. (1987) "Two-dimensional finite bin-packing algorithms." *Journal of the operational research society* **38(5)**: 423-429.

[12] Lodi, Andrea, Silvano Martello, and Daniele Vigo. (1999) "Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems." *INFORMS Journal on Computing* **11(4)**: 345-357.

[13] Ahmed, Ben Mohamed, (2009) "Résolution approchée du problème de bin-packing", Ph. *D. thesis*, Le Havre.

[14] Gonçalves, José Fernando, and Mauricio GC Resende. (2013) "A biased random key genetic algorithm for 2D and 3D bin packing problems." *International Journal of Production Economics* **145(2)**: 500-510.

[15] Soke, Alev, and Zafer Bingul. (2006) "Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems." *Engineering Applications of Artificial Intelligence* **19(5)**: 557-567.

[16] Parreño, Francisco, et al. (2010) "A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing." *Annals of Operations Research* **179(1)**: 203-220.

[17] Boschetti, Marco A., and Vittorio Maniezzo. (2005) "An ant system heuristic for the two-dimensional finite bin packing problem: preliminary results", In *Multidisciplinary Methods for Analysis Optimization and Control of Complex Systems*, Springer, Berlin, Heidelberg.

[18] Shin, Young-Bin, and Eisuke Kita. (2017) "Solving two-dimensional packing problem using particle swarm optimization." *Computer Assisted Methods in Engineering and Science* **19(3)**: 241-255.

[19] Askarzadeh, Alireza. (2016) "A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm." *Computers & Structures* **169**: 1-12.

[20] Kennedy, James, and Russell C. Eberhart. (1997) "A discrete binary version of the particle swarm algorithm", In *Proceedings IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation* **5**: 4104-4108, Piscataway.

[21] Falkenauer, Emanuel, and Alain Delchambre. (1992) "A genetic algorithm for bin packing and line balancing", *In Proceedings IEEE International Conference on Robotics and Automation*, May 10–15, Nice, France.