Int J Adv Manuf Technol (2015) 81:455–464
DOI 10.1007/s00170-015-7107-1

ORIGINAL ARTICLE

A GRASP meta-heuristic for two-dimensional irregular cutting stock problem

S. A. MirHassani[1] · A. Jalaeian Bashirzadeh[1]

Received: 11 October 2014 / Accepted: 30 March 2015 / Published online: 9 May 2015

Abstract Reducing expensive raw material waste is an important goal in the industry. In this paper, two-dimensional irregular cutting stock problem—a nesting problem that differs from other in their irregular shape of the pieces—with demand is studied, in which the required pieces has to be produced from large rectangular sheet minimizing material waste. Structure of this problem made it intractable for practical applications such that exact algorithms are not able to solve it in a reasonable time. Greedy randomized adaptive search procedure (GRASP) meta-heuristic algorithm is adapted to tackle the problem by providing high-quality solution in an appropriate time. The algorithm does not depend on the shape (convexity and regularity) of pieces and is able to deliver an optimum solution for instances up to 30 pieces of 7 different types. In addition, computational results are provided for different test problems from the related literature.

Keywords Two-dimensional cutting stock problem · Nesting problem · Irregular cutting stock problem · GRASP procedure

1 Introduction

Cutting stock problem (CSP) belongs to a specific category of problems called cutting and bin-packing problems. In this sense, the CSP is one of the well-known problems in the operation research that can be an effective way in minimizing material waste. Material waste is a phenomenon of mass production, and often, industries are searching some ways to prevent and interrupt the trend. According to many applications of CSP in glass, steel, wood, space, and apparel industry, it can be concluded that one of the solutions is to employee the CSP [25] (also see [35] among the others).

In the CSP, small pieces are cut from the large stocks so that material waste is minimized. In addition, quality of materials is also playing an important role in some industries such as textile and apparel. Every CSP can be divided into two main parts: the first part is cutting area, in which the pieces are placed or cut such as bars, sheets, fabric, wood, etc. The second part is small pieces that are required and will be cut from the large stocks.

Since the 1950s that computers emerged as fast processors, the CSP has been more extensively considered and its response will have significant practical and industrial importance. Over the years, two main solution methods have appeared: heuristic approaches and approximations based on linear programming.

Dimensions (one, two, and three), pattern shapes (regular or irregular), and number of original board are the main features of grouping the problem [14]. Given that, it is an NP—hard problem, and researchers have tended to use heuristic approaches that create near-optimal solutions in lieu of achieving the optimal patterns. In previous works, most of the discussed problems have only considered large rectangular sheets and irregular shapes to a lesser degree. So, it seems that further investigations are required in this regard.

Many studies on two-dimensional cutting stock problems (2DCSP) have been devoted to the guillotine cuts. Simply stated, in guillotine cut, cutting occurs in a large rectangular sheet from one edge to face another. Formulating the problem in higher dimensions is similar to the one-dimensional case. In this case, the definition and generation of patterns are more complicated. The simplest one occurs when the board or

✉ S. A. MirHassani
a_mirhassani@aut.ac.ir

A. Jalaeian Bashirzadeh
Al.Jal@aut.ac.ir

[1] Department of Applied Mathematics, Amirkabir University of Technology, Tehran, Iran

material and small pieces are rectangular. In the two-dimensional irregular cutting stock problem (2DICSP), the required pieces with irregular shapes (polygon, non-convex, and curve) should be produced from board or large rectangular sheet minimizing material waste. In [34], a good collection of all kinds of methods was presented to solve the problem. The methods based on the approximation and analysis were commonly used to reduce the problem complexity.

Nesting problem belongs to the more generic class of 2DICSPs. These problems are not only a scientific challenge, as they add the geometry handling complexity to the combinatorial optimization nature of CSP, but they are also most relevant in real-world applications [33].

1.1 Literature review

The CSP was first raised in 1939 by Kantorovich who reflects the history and importance of the problem over the past years. He investigated the problem in order to organize the lumber and optimally use materials and began to formulate a linear programming problem. More research on this problem goes back to around 1960 to 1980. Among the most well-known studies, the research by Gilmore and Gomory can be mentioned. The most common approach used for this problem is linear programming. Gilmore and Gomory solved the problem in 1961 as a linear programming problem (Gilmore and Gomory, A linear programming approach to the cutting stock problem, [18]) and (Gilmore and Gomory, A linear programming approach to the cutting stock problem part II, [19]). Their objective was to minimize the cost, and a large number of patterns and columns were identified as a weakness of their methods. Later, their idea of column generation encouraged many researchers to use the method. In 1964, Pierce was one of those who used it in the paper industry to reduce material waste [27]. Hahn [20] posed the problem with dynamic programming as the large rectangular sheet, which contained some defects [20]. Sarker [32] applied the same approach to solve one-dimensional cutting problem [32]. Hahn solved a three-stage guillotine CSP, in which large rectangular sheets had different qualities in different areas and used the dynamic programming method to solve the maximization of values on the large rectangular sheet [35]. Beasley extended the problem to the n-stage and unconstraint guillotine state and used recursive dynamic programming by the idea of *normal patterns* to reduce computational time [6]. Non-guillotine idea was first proposed in 1984 by Biro and Baros [9]; using graph theory, they introduced an idea for non-guillotine patterns, not aiming to achieve optimal solutions. In 2003, Kalvelagen implemented column generation method, by which he could count all the possible patterns for the problem [35]. In 2007, Reinaldo and Luciano adapted another idea to produce patterns that had constraints on longitudinal and cross guillotine and 180° rotation cutting; their paper is based on the recursive dynamic programming method and greedy constructive algorithm [26]. Arbib and Marinelli [5] investigated a waste minimization problem in the glass industry where the cutting process was performed in two phases, and waste reduction problem was considered in both phases [5]. In 2009, Yaodong and Yiping investigated rectangular 2DCSP in the industry of bridge-building and took advantage of heuristic algorithm for solving the problem [36]. Haims and Freeman [21] presented a problem called “template layout” by converting irregular shapes into rectangular shapes and then solving the problem as a multi-stage problem [21]. Albano presented a semi-automatic system to produce the optimal layout of irregular shapes on a large rectangular sheet; his system was called “computer-assisted layout generation system” [2]. Arbel discussed cutting timing based on the column generation method [4]. A heuristic search idea was adapted by Albano and Sapuppo [3] for the placement of pieces with irregular shapes inside the large rectangle sheet so that all irregular pieces were placed and length of the placement was minimized. Roberts [31] conducted a study on L-shape forms [11], which have the ability to be paired with each other. Another idea for the CSP with irregular shapes was considered by Qu and Sanders [30]. Prasad and Dhande followed the issue in using CAD software and provided some software in this regard [29]. CAD systems have applications in the apparel industry for cutting stocks with irregular shapes and specific designs. In addition, nesting problem has been examined by many authors and researchers; however, given the understanding of the problem complexity, none of the authors have provided a precise mathematical programming model and a suitable solving algorithm for guaranteeing the optimal solution. Adamowicz and Albano [1] used an idea and geometric method to optimize the placement of irregular shapes inside a rectangle using the concept of no-fit polygon (NFP) [1]. At the same time, recent approaches with better performances were introduced by Egeblad et al. [15], Bennell and Song [8], and Leung et al. [23]. In recent years, the general ideas for investigating nesting problems were presented by Bennell and Oliveira [7]. In addition, Toledo et al. [33] adapted a mixed integer model where the binary decision variables corresponded to dotted-board and piece types [33]. A major achievement of this model is its lack of sensitivity to board geometry and piece types that can make it applicable to more complex problems such as non-convex boards with defects. Moreover, the number of binary variables did not depend on the total number of pieces, but depended on the number of different types of pieces, which facilitated solving the problems with a smaller variety of pieces. They studied 45 problem instances with the new model, solving to optimality 34 instances with various total numbers of pieces and get the optimum solution for cases with 7 types and 21 total numbers of pieces. By the same token, this method does not have a pleasant scalability. More detailed description of nesting problems and cutting and packing problems can be found in [13]. Over the last 10 years, most efforts have been concerned with the improvement of meta-heuristic algorithms using local searches. For example, Tabu Search by Blazewicz et al. [10] can be noted [10]. Lutifiyya et al. solved ICSP using simulated annealing [24]. For genetic algorithms, Ismail and Hon [22] can be mentioned [22]. Many heuristic and meta-heuristic approaches have been used for this problem. For further details, see the related literature [35].

The main contribution of this paper is to adapt greedy randomized adaptive search procedure (GRASP) and reactive GRASP algorithms for solving 2DICSP through dotted-board model proposed by Toledo et al. in [33]. This work reveals the efficiency and performance of algorithms without any coherency to the shape and structure of the pieces. In conjunction with that, according to the considered instances with varied piece types, it achieved a quiet appropriate computational time in comparison to other methods, besides, placing the maximum number of pieces on a board. Furthermore, these algorithms were able to deliver an optimum solution for instances up to 30 total number of pieces of 7 different types in appropriate time. In this case, the required pieces with irregular shapes should be produced from large rectangular sheets or boards so that the board length is minimized.

This paper is organized as follows. In Section 2, the dotted-board model proposed by Toledo et al. [33], which is a new mixed integer programming model, is briefly introduced. Then, in Section 3, the GRASP algorithm is fully introduced and adapted to solve the 2DICSP problem. Afterward, reactive GRASP algorithm is introduced in Section 4. The computational results and discussion of the algorithm's performance are presented in Section 5. In the last section, some conclusions from the research output and their limitations are reported.

2 Mathematical model

In nesting problem, the distance between two pieces is calculated based on their coordinates. The most common way for displaying irregular shapes is the use of a piece or shape in the form of polygon. In this study, all pieces or shapes were considered as polygons and pieces were not allowed to rotate.

In this section, first, the model of 2DICSP taken from the study by Toledo et al. is introduced. The dotted-board model for the nesting problem (2DICSP) can be expressed as follows:

$$\min \quad \max_{t\in\tau, d\in \mathrm{IFP}_t}\left(c \times g_x + x_t^M\right) \times \delta_t^d \tag{1}$$

Subject to

$$\sum_{d\in \mathrm{IFP}_t} \delta_t^d = q_t \quad , \qquad \forall t\in\tau; \tag{2}$$

$$\delta_u^e + \delta_t^d \le 1 \qquad \forall e\in \mathrm{NFP}_{t,u}^d, \quad \forall t,u\in\tau, \quad \forall d\in \mathrm{IFP}_t; \tag{3}$$

$$\delta_t^d \in \{0,1\} \qquad \forall d\in \mathrm{IFP}_t, \quad \forall t\in\tau; \tag{4}$$

where δ_t^d is a decision variable for a pair (dot and piece type). It is equal to 1 if the reference point of piece type t is located at dot d; otherwise, 0. Objective function (1) minimizes the maximum length of the board. Constraint (2) ensures that pieces demand are met, constraint (3) reflects the lack of overlap between the pieces, and constraint (4) defines the decision variables only for the dots in which the pieces are inside the board. The board is considered to be a rectangle with fixed width W (board width for each instance is fixed). Also, an upper bound L is defined.

For each piece, a reference point in one of its corners is considered. By definition of set $\mathrm{NFP}_{t,u}$ between two pieces t and u, we conclude that if the reference point of piece u is located inside $\mathrm{NFP}_{t,u}$, then the two pieces overlap and interfere. If the reference point of piece u is on the boundary line of $\mathrm{NFP}_{t,u}$, the two pieces will be in contact with each other; finally, if the reference point of piece u is located outside $\mathrm{NFP}_{t,u}$, the two pieces will not be in contact. Here, t and u are indicative of pieces $t,u\in\tau$; $\tau=\{1,\ldots,T\}$, c shows the column of board, $c\in\mathcal{C}$; $\mathcal{C}=\{0,\ldots,7C-1\}$, d represents the dots in board, $d\in\mathfrak{D}$; $\mathfrak{D}=\{0,\ldots,D\}$, g_x, g_y: grid resolution on the x-axis and y-axis, C: number of columns (L/g_x+1), R: number of rows (W/g_y+1), D: total number of dots on board ($C\times R$), and q_t is the demand of piece type t ($t\in\tau$).

IFP concept for a piece in relation to a board shows dots on the board in which if the reference point of the piece type t is located, then the entire piece will be inside the board [33].

3 GRASP algorithm

Today, with large-scale problems and the importance of quick solutions, classic algorithms cannot solve many problems, and thus, random search algorithms are more utilized. A meta-heuristic optimization algorithm with small changes can be used for a variety of optimization problems. The important point is to use some mechanisms for exiting local optimum points. One of these algorithms is the so-called *greedy randomized adaptive search procedure* (GRASP).

Feo et al. use the GRASP for the first time in 1989 [16] to solve set covering problem. The GRASP is a multi-start meta-heuristic where each iteration consists of two phases of construction and local search. The construction phase builds a feasible solution by using a greedy randomized algorithm. The local search phase works in an iterative fashion by successively replacing the current solution with a better one in the neighborhood and ends when no better solution is found in the area. The best overall solution is kept as the GRASP result. In Fig. 1, the overall pseudo-code of the GRASP algorithm for minimization problem is detailed [17].

3.1 Implementing GRASP for 2DICSP

In this section, a GRASP algorithm is described for the 2DICSP based on dotted-board model. The algorithm is divided into two main parts:

The first part is adapted as the algorithm preprocessing and provided appropriate data for the main part. The second part begins with implementing the GRASP algorithm by using the data generated in the first part.

3.1.1 Preprocessing

In this subsection, three main parameters of the problem as input to the algorithm are presented: defining the required pieces and board dimensions including columns (length) and rows (width). Implementation of this part is done in AIMMS 3.12 software.

Required pieces according to the model are defined as follows: If a board with C columns and R rows are given, then the dotted-board contains $C\times R$ dots. Each piece has a reference point for defining the piece on the dotted-board. The reference points of each piece are defined on the bottom left corner. An example of a reference point is given in Fig. 2. The highlighted points are reference points. If a part of the piece is located inside the square, the point at the bottom left of the square is activated. If a piece uses one square, then another piece cannot use the same square.

In Fig. 3, the activated point of two pieces is depicted as an example. In fact, each highlighted point means that the square on the right and upward is used by the desired piece.

```
Procedure GRASP(Stop_Iteration, Seed)
    1.  Read Input;
    2.  For k=1,…, Stop_Iteration do
    3.  Solution ← GreedyRandomizedConstrucion (Seed);
    4.  If Solution not feasible do
    5.  Solution ← Repair (Solution);
    6.  End If;
    7.  Solution ← LocalSearch(Solution);
    8.  UpdateSolution(Solution, BestSolution);
    9.  End For;
    10. Return BestSolution
End GRASP
```

Fig. 1 Pseudo-code of the GRASP meta-heuristic

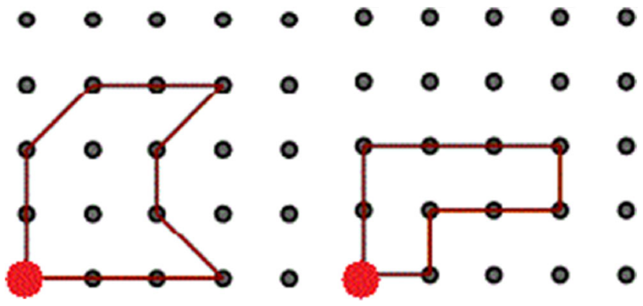

Fig. 2 Reference point definition

The most important issue in the preprocessing section is the lack of overlap of pieces. For each piece, the corresponding zero-one matrix is formed in the dotted-board. In order to examine the existence of overlap between two pieces, their zero-one matrixes are compared. In fact, if the pieces will overlap at the specific dots, then the overlap condition (3) must be considered for two pieces.

So, if piece type t_1 is located in row i_1 and column j_1 and piece type t_2 is located in row i_2 and column j_2 and the pieces overlap, the corresponding matrix element is 1; otherwise, 0. The overlapped dots are identified for all the pieces. Also, a set of dots is generated and used as the *feasible points* in the main part.

The overlap between each two pieces is modeled through matrix $T_{i,j}$. In this matrix, there is only two non-zero element in each row corresponding to a conflict between two objects which leads to a constraint. For example, if there are 4 piece types and a board with 6 rows and 10 columns, then 240 variables will exist. Also, if the number of overlap between all pieces (dots) is equal to 1000, a matrix with the size of 1000×240 will be generated. For instance, the matrix $T_{i,j} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$ is a binary constraint matrix corresponding to two constraints of $x_1+x_2\leq1$ and $x_2+x_3\leq1$.

But, given that the CSP parameters $G_{t,c,r}$ are trio (piece types and column and row for reference point), respectively, the GRASP algorithm requires a single-

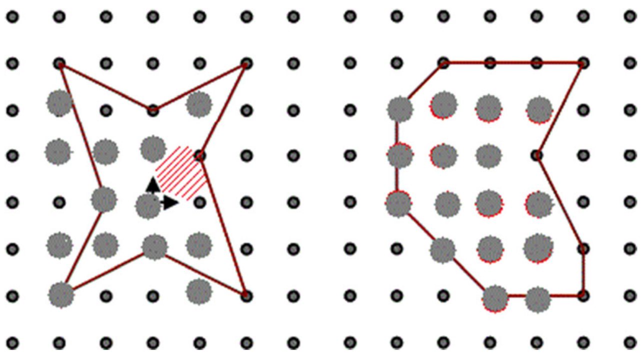

Fig. 3 Point activation of two pieces

```
1. $I \leftarrow C \times R \times T$
2. $x_j \leftarrow 0, \forall j \in J$
3. $Evalj \leftarrow 1/\sum_i T_{i,j}$ , $\forall j \in J$
4. $While\ (I \neq \emptyset)\ do$
5.    Limit$\leftarrow min_{j\in J}(Evalj) + \alpha * (max_{j\in J}(Evalj) - min_{j\in J}(Evalj))$
6.    RCL$\leftarrow \{ j \in J,\ Evalj \geq Limit \}$
7.    Do
8.       $i^* \leftarrow RandomSelect(RCL)$
9.       $RCL \leftarrow RCL - \{i^*\}$
10.      $I \leftarrow I - \{i^*\}$
11.   $While$ ( Condition ( $i^*$ ) )
12.   End $While$
13.   $x_{i^*} \leftarrow 1$
14.   $\begin{cases} demand\left(\left\lfloor i^*/_{C\times R}\right\rfloor + 1\right) - 1 & if\ i^*/_{C\times R} \notin \mathbb{Z} \\ demand\left(i^*/_{C\times R}\right) - 1 & if\ i^*/_{C\times R} \in \mathbb{Z} \end{cases}$
15.   $I \leftarrow I - \{j : \exists\, i, T_{i,j} + T_{i^*,j} > 1\}$
16. End$While$
```

Fig. 4 Pseudo-code of the greedy randomized construction phase for the 2DICSP

component vector x_j. So, using map (7), the three components t, c, and r should be converted into a single-component j and the main part of the algorithm should be implemented according to the one-dimensional vector x with component x_j.

$$j = (C \times R) \times (t-1) + R \times (c-1) + r \tag{7}$$

Matrix $T_{i,j}$ is used to generate binary constraint set $\sum_i T_{i,j} x_j \leq 1$.

3.1.2 The GRASP algorithm

To implement the algorithm, it should be taken into consideration that two important factors: lake of overlap between pieces and the demand, should be met while using the minimum length of the board. In addition, for the construction and local search phase and also an examination of the lack of overlap between the pieces, the work is reported by Delorme et al. is used [12].

3.1.3 Greedy randomized construction

The algorithm inputs are the number of columns and rows of the board, number of piece types, demand for each piece, the GRASP parameter, α, and total number of iterations for the algorithm as well as local search.

Problem solution vector consists of components as x_j, $j \in J = \{0, 1, \ldots, T \times C \times R\}$ (C is column, R is row, and T is piece types). So, any variable represents a dot of the board and is corresponding to a particular piece. In fact, every component of vector x at the end converts into a trio $G_{t,c,r}$ based on a mapping (7). According to the pseudo-code of Fig. 4, first, the set, which is equal to total number of binary variables of the problem, is equalized to the product of the multiplication of the number of columns by the number of rows times the number of piece types (*set I*). In line (2), all decision variables are given the initial values and it starts from a feasible solution. In line (3), a greedy function called *Evalj* is used so that the variables are appearing in a fewer number of binary constraints (overlaps) enjoy a higher priority of selection. *Evalj* controls the greedy aspects of the algorithm and plays a significant role in the construction phase. From lines (4) to (16), the loop of the procedure is performed and a complete solution for the problem is produced. Lines (5) and (6) show how to construct the list of best variables called the restricted candidate list (RCL) based on two important factors of *Evalj* and *Limit*. For the algorithm parameter, α, we considered fixed value, close to the purely greedy and random choice according to [17]. In fact, as the greedy function of variables is greater than or equal to the *Limit*, the variables are good choices to have value 1. Lines (7) to (12) are the random selection of a i^* variable from RCL to set to 1.

Condition term considered in line (11) meanings:

1. i^* is corresponding to *feasible points* on board.

Fig. 5 Pseudo-code of the local search for the 2DICSP

```
1. Function$SearchNeighbourhood(s, N)$
2. Find$\acute{s} \in N(s)$
3. $s \leftarrow \acute{s}$
4. Return$s$
5. End$SearchNeighbourhood$
6. While( Iteration > 0)do
7. Solution$\leftarrow$ $SearchNeighbourhood$( Solution, 0-1 exchange )
8. Solution$\leftarrow$ $SearchNeighbourhood$( Solution, 1-1 exchange )
9. Solution$\leftarrow$ $SearchNeighbourhood$( Solution, 1-2 exchange )
10. Solution$\leftarrow$ $SearchNeighbourhood$( Solution, 2-2 exchange )
11. Solution$\leftarrow$ $SearchNeighbourhood$( Solution, 0-n exchange )
12. Iteration$\leftarrow$ Iteration$-1$
13. EndWhile
14. Return Solution
```

2. Demand of corresponding piece has not satisfied yet.

The line (14) updates the number of pieces that currently are on board. Finally, all variables that are in conflict with x_i– are set to 0. The algorithm continues until the value of all variables fixed to 0 or 1 and RCL get empty.

3.1.4 Local search phase

The neighborhood definition is based on the concept of *k-p* exchange. A *k-p* exchange is generating a new solution based on *x* where the value of *k* variables is varying from 1 to 0 and the value of *p* variables is changing from 0 to 1.

Only three following characteristics are considered for this part:

1. Due to the number of different combinations in the *k-p* exchange, only exchanges of (0-1), (1-2), (1-1), and (2-2) are used; however, one can employ more combinations; but, definitely time will increase as well.
2. At the end, the 0-*n* exchange is used; if none of the two variables in a binary constraint take 1, one of them set to 1 randomly and then the local search process continues.
3. Finally, after each exchange, two conditions mentioned in the construction phase must be satisfied.

The local search restarts from the new solution and stops with the maximum iteration. In Fig. 5, the pseudo-code for the local search is reported.

Eventually, at the end of the construction phase and local search, a solution vector is obtained that shows the minimum length without overlapping pieces while demands are also met.

4 Reactive GRASP

As stated in the description of the GRASP algorithm, the use of an alternative method for improving the GRASP may have some advantages. One of which is adjustment of the parameter

```
1. Solution← ∅;
2. ProbAlpha ← $1/|alphset|$, $\forall \alpha \in alphaset$;
3. While ( Iteration > 0 ) do
4. $\alpha^*$ ← RandSelect ( alphaset, Probalpha ) ;
5. firstSol ← $GreedyRandomized\ (\alpha^*)$ ;
6. $improveSol \leftarrow localSearch\ (firstSol)$ ;
7. Solution←Solution∪ {$improvedSol$} ;
8. If$ProbaUpdate = True$ do
9. $val_\alpha \leftarrow \left(\frac{mean(z(s))-z(Worst(Solution))}{z(Best(Solution))-z(Worst(Solution))}\right)$, $\forall \alpha \in alphaset$;
10. $ProbAlpha \leftarrow \frac{val_\alpha}{(\sum_{\alpha^* \in alphaset} val_{\alpha^*})}$, $\forall \alpha \in alphaset$;
11. EndIf
12. $End$While
13. $Return\ Best\ (Solution)$;
```

Fig. 6 Pseudo-code of reactive GRASP for the 2DICSP

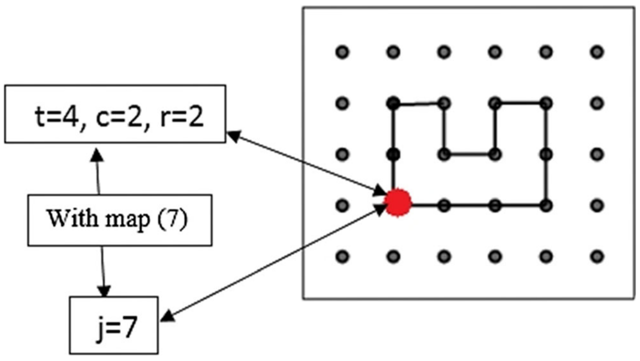

Fig. 7 Conversion of three dimension with single-dimension vector and vice versa

automatically based on the quality of the solutions. In this case, the value of the RCL parameter, α, is not fixed, but instead is selected at each iteration from a discrete set of possible values. This selection is associated by the solution values found along the previous iterations. In Fig. 6, *reactive GRASP* or *RGRASP* is used to avoid parameter adjusting manually. First, the set of different numbers in the range [0, 1] is chosen and placed in a set called *alphaset*. Another parameter is *ProbAlpha* which represents the selection probability of each α in different iterations. Before beginning the iterations, the selection probability of all values is equal. New possibilities based on the condition *ProbaUpdate* are calculated in lines (9) and (10).

This idea is taken from the work by Prais and Ribeiro [28] as well as Delorme et al. [12]. In this algorithm, we are generally seeking for a parameter value (between 0 and 1) that produces a better solution.

Table 1 The details for 2DICSP instances

Instances	Total number of pieces	Number of dots	Number of binary variables	Total number of constraint
RCO1	7×1	144	1008	35,630
RCO2	7×2	256	1792	110,586
RCO3	7×3	368	2576	185,542
RCO4	7×4	480	3360	260,498
RCO5	7×5	608	4256	346,162
BLZ1	7×1	144	1008	34,124
BLZ2	7×2	240	1680	95,000
BLZ3	7×3	352	2464	155,876
BLZ4	7×4	480	3360	237,044
BLZ5	7×5	640	4480	306,969
SHAPES2	8	615	2460	43,079
SHAPES4	16	1148	4592	466,466
SHAPES5	20	1353	5412	612,271
SHAPES7	28	2009	8036	1,109,251
SHAPES9	34	2255	9020	1,831,490
SHAPES15	43	2788	11,152	1,928,374

Table 2 Computational results for GRASP algorithm

Instances	# of pieces	Best obj. val.	Best solution with GRASP				Average time GRASP(s)	Gap (%)
			α=0.4	α=0.6	α=0.7	α=0.9		
RCO1	7	8*	8	8	8	8	1.03	0
RCO2	14	15*	17	16	15	16	5.67	0
RCO3	21	22*	22	23	22	22	14.5	0
RCO4	28	29	30	30	29	28	28.25	−3.4
RCO5	35	37	40	39	38	39	43.92	2.7
BLZ1	7	8*	8	8	8	8	0.909	0
BLZ2	14	14*	15	15	15	16	4.78	7.1
BLZ3	21	20*	23	23	22	23	13.8	10
BLZ4	28	28	31	31	30	31	27.95	7.1
BLZ5	35	35	37	37	36	37	40.3	2.8
SHAPES2	8	14*	15	15	14	14	3.89	0
SHAPES4	16	25*	28	27	27	27	42	8
SHAPES5	20	30	33	33	33	33	79.34	10
SHAPES7	28	45	49	48	47	47	240.56	4.4
SHAPES9	34	54	60	59	57	58	301.67	5.55
SHAPES15	43	54.76	61	59	58	59	311.12	5.9

* Optimum solution

Finally, using (7), solutions x_j are mapped into a trio $G_{t,c,r}$ and plotted. In Fig. 7, a view of this mapping is depicted.

5 Computational results

To evaluate the algorithm performance on 2DICSP, three groups of instances taken from the Toledo et al. [33] are used which a brief description of them is presented. *BLZ* group includes non-convex pieces and *RCO* group includes convex pieces. The number of piece types in both *RCO* and *BLZ* instances is seven. *RCOn* and *BLZn* mean that *n* is the demand number for each piece which ranges from 1 to 5. For these two groups, the board width is considered W=15. The third group that has larger dimensions is *SHAPES*. This group includes a variety of four pieces and is provided in six groups with different demands

Table 3 Computational results for RGRASP algorithm

Instances	# of pieces	Best obj. val. (seen)	Best obj. val. (RGRASP)	Best α	Gap (%)
RCO1	7	8*	8*	0.7	0
RCO2	14	15*	16	0.8	6.6
RCO3	21	22*	22*	0.9	0
RCO4	28	29	27	0.8	6.8
RCO5	35	37	38	0.7	−2.7
BLZ1	7	8*	8*	0.8	0
BLZ2	14	14*	14*	0.7	0
BLZ3	21	20*	20*	0.8	0
BLZ4	28	28	30	0.7	7.1
BLZ5	35	35	35	0.75	0
SHAPES2	8	14*	14*	0.9	0
SHAPES4	16	25*	28	0.9	12
SHAPES5	20	30	32	0.8	6.6
SHAPES7	28	45	47	0.7	4.4
SHAPES9	34	54	55	0.75	1.8
SHAPES15	43	54.76	57	0.8	4

* Optimum solution

Table 4 The best results of both GRASP and RGRASP algorithms

Instances	# of pieces	Best obj. val. (published)	Best obj. val. (obtained)	GRASP or RGRASP	Best α
RCO1	7	8*	8*	GRASP and RGRASP	0.7
RCO2	14	15*	15*	GRASP	0.7
RCO3	21	22*	22*	GRASP and RGRASP	0.9
RCO4	28	29	27	RGRASP	0.8
RCO5	35	37	38	GRASP and RGRASP	0.7
BLZ1	7	8*	8*	GRASP and RGRASP	0.8
BLZ2	14	14*	14*	RGRASP	0.7
BLZ3	21	20*	20*	RGRASP	0.8
BLZ4	28	28	30	GRASP and RGRASP	0.7
BLZ5	35	35	35	RGRASP	0.75
SHAPES2	8	14*	14*	GRASP and RGRASP	0.9
SHAPES4	16	25*	27	GRASP	0.7
SHAPES5	20	30	32	RGRASP	0.8
SHAPES7	28	45	47	GRASP and RGRASP	0.7
SHAPES9	34	54	55	RGRASP	0.75
SHAPES15	43	54.76	57	RGRASP	0.8

* Optimum solution

and board width of W=40. The algorithms are coded using C# 2013 programming software implemented on *MSI* laptop, 4 GB of RAM memory, a 2.2-GHz processor.

The algorithm presented in Section 3 is implemented on the basic instances. These instances include 7 to 43 pieces. Table 1 shows the characteristics of the instances where the first column shows the problem name. The total number of pieces, dots, variables (multiplying the number of dots by the total number of pieces), and constraints (total overlaps) are shown in columns 2, 3, 4, and 5, respectively.

The algorithm run for all instants in Table 1 under different parameter values $\alpha=\{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$. The results for the best parameters are reported in Table 2. For all instances, the number of iterations for the local search and the GRASP algorithm were 10 and 200, respectively.

In Table 2, the values marked with an asterisk are corresponding to the optimum value and the other shown the best value obtained by Cplex12 with the time limit of 18,000 s reported by Toledo et al. in [6]. The fourth column consists of the best solutions found under different parameters. The average time and the gap between two approaches are reported in the last two columns.

According to Table 2, the algorithm was able to find the optimum solution for five out of eight instances (62.5 % success). For one instance, the best objective value found is better than the best published result (RCO4). Average gap for all instances is equal to 3.75 % while the average time of GRASP algorithm is 72.48 s, indicating that the algorithm is able to find good near-optimal solution quickly.

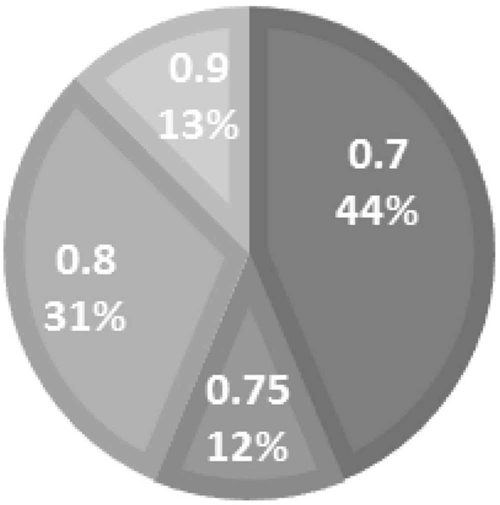

Fig. 8 The rate of success of the different values of parameter α

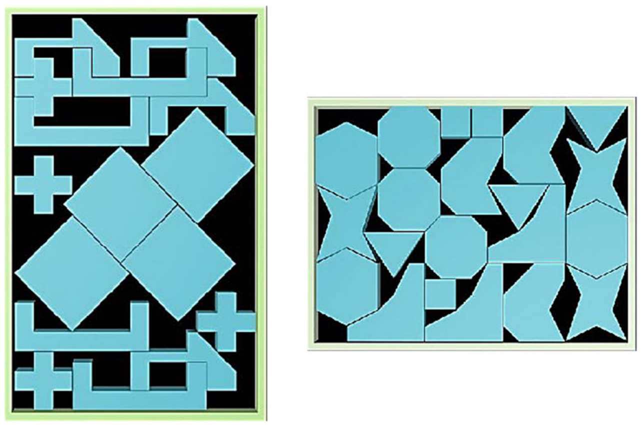

Fig. 9 *Right* Optimal layout for instance BLZ3. *Left* Optimal layout for instance SHAPES4

In Table 3, the results obtained by the RGRASP algorithm for the same instances are given. For all instances, the total number of iterations is equal to 250 and the *alphaset*={0.1, 0.3, 0.4, 0.6, 0.65, 0.7, 0.75, 0.8, 0.9, 1}. *ProbaUpdate* condition is applied for every 10 iterations. As mentioned in Section 4, α automatically is calculated in the RGRASP algorithm.

The RGRASP algorithm was able to find the optimum solution for six out of eight instances (75 % success); in one instance, the best objective value found by the RGRASP algorithm is better than the best published result. Average gap for all instances is equal to 2.4 %. The best performance is for α from 0.7 to 0.9.

Table 4 summarizes the best results of both GRASP and RGRASP algorithms for all the instances. In Table 4, the fourth column records the results by either GRASP or RGRASP algorithms and the fifth column is the best objective value; in the last column, the best α corresponding to the best solution is reported.

So, for the instances with 35 pieces, the best published solutions are achieved; for the case of 28 pieces, an improvement is observed. Overall, the RGRASP in 87.75 % and the GRASP in 56.25 % cases were able to find the best solutions. In terms of execution time, since the GRASP algorithm lacks memory, thus, it has better computational time.

In Fig. 8, the rate of success of the different values of parameter α is depicted.

Overall, for 16 instances, in 9 cases, the proposed algorithms led to the best released solutions and, for the rest, were close to the best published results with a 2.4 % gap.

According to the results, by increasing the size of board, the algorithm's performance will be affected, which is not too far-fetched. Although the number of decision variables is not related directly to the number of pieces, the larger number of pieces makes the board bigger and indirectly increases the number of variables. For this reason, solving the instances of *SHAPES* group is more difficult. Although in this group, the variety of pieces is lower than other groups; because of larger pieces, the number of variables and total iterations of the algorithm are increased. The algorithm does not depend on the shape of pieces and is able to deliver an optimal solution for the instances of up to 30 pieces. However, the computational results indicate the relative superiority of the algorithm in solving the instances of convex (four out of five instances in the *RCO* group) and non-convex (four out of five instances in the *BLZ* group) shapes. In Fig. 9, some examples of optimized instances are drawn.

6 Conclusion and further works

In this paper, a new GRASP/reactive GRASP algorithm was designed for the 2DICSP and the total number of pieces was increased to the range of 7 to 43. These algorithms build up the discretization of the board, producing a grid of dots that become the feasible positioning points for the pieces. In addition, the performance of the algorithm was evaluated on basic instances. Computational results showed the capability and effectiveness of the algorithms. Generally, for 9 out of 16 instances, the best published solutions were accomplished and were close to the best solutions for other instances with a 2.4 % average gap. In addition, best values for the algorithm parameter with regard to the problem were calculated. Although the RGRASP algorithm increased the solution time naturally, the quality of the solutions was also improved. In addition, the strength was the independence of proposed algorithms to shape of the pieces (regularity and convexity); moreover, it was able to achieve an optimal solution for the instances of up to 30 total numbers of pieces with 7 piece types at a favorable computational time. Finally, we would like to point out that the algorithm is quite flexible and could be adapted to accommodate other conditions or constraints. In addition, for the better performance in local search, it can make use of other heuristic algorithms such as Tabu search. In the construction phase, a more suitable standard, in which *Evalj* was built, can be considered to improve the quality of the solutions. Moreover, the use of other preprocessing phases to reduce the problem size in terms of number of variables and constraints is valuable.

References

1. Adamowicz M, Albano A (1976) Nesting two dimensional shapes in rectangular modules. Comp Aid Des 8(1):27–33
2. Albano A (1977) A method to improve two-dimensional layout. Comp Aid Des 9(1):48–52
3. Albano A, Sapuppo G (1980) Optimal allocation of two-dimensional irregular shapes using heuristic search methods. IEEE Trans Syst Man Cybern, SMC 10(5):242–248
4. Arbel A (1993) Large-scale optimization methods applied to the cutting stock problem of irregular shapes. Int J Prod Res 31(2):483–500
5. Arbib C, Marinelli F (2007) An optimization model for trim loss minimization in an automotive glass plant. Eur J Operat Res 183(3):1421–1432
6. Beasley J (1985) Algorithms for unconstrained two-dimensional guillotine cutting. Operat Res 36(4):297–306
7. Bennell JA, Oliveira JF (2009) A tutorial in irregular shape packing problems. J Operat Res Soc 60:93–105
8. Bennell J, Song X (2010) A beam search implementation for the irregular shape packing problem. J Heurist 16(2):167–188
9. Biro M, Boros E (1984) Network flows and non-guillotine cutting patterns. Eur J Operat Res 16(2):215–221
10. Blazewicz J, Hawryluk P, Walkowiak R (1993) Using a tabu search approach for solving the two-dimensional irregular cutting problem. Annal Operat Res 41(4):313–325
11. Cheng C, Feiring B, Cheng T (1994) A cutting stock problem—a survey. Int J Prod Econ 36(3):291–305
12. Delorme X, Gandibleux X, Rodriguez J (2004) GRASP for set packing problems. Eur J Operat Res 153(3):564–580

13. Dowsland K, Dowsland W (1995) Solution approaches to irregular nesting problems. Eur J Operat Res 84(3):506–521
14. Dyckhoff H (1990) A typology of cutting and packing problems. Eur J Operat Res 44(2):145–159
15. Egeblad J, Nielsen B, Odgaard A (2007) Fast neighborhood search for two- and three-dimensional nesting problems. Eur J Operat Res 183(3):1249–1266
16. Feo T, Resende A (1989) A probabilistic heuristics for a computationally difficult set covering problem. Operat Res Lett 8:67–71
17. Gendreau M, Potvin J (2010) Handbook of metaheuristics, 2nd edn. Springer, New York
18. Gilmore P, Gomory R (1961) A linear programming approach to the cutting stock problem. Operat Res 9(6):849–859
19. Gilmore P, Gomory R (1963) A linear programming approach to the cutting stock problem part II. Operat Res 11(6):863–888
20. Hahn S (1968) On the optimal cutting of defective sheets. Oper Res 16(6):1100–1113
21. Haims M, Freeman H (1970) A multistage solution of the template-layout problem. IEEE Trans Syst Sci Cybernet 6(2):145–151
22. Ismail H, Hon K (1992) New approaches for the nesting of two-dimensional shapes for press tool design. Int J Prod Res 30(4):825–837
23. Leung S, Lin Y, Zhang D (2012) Extended local search algorithm based on nonlinear programming for two-dimensional irregular strip packing problem. Comp Operat Res 39(3):678–686
24. Lutfiyya H, McMillin B, Poshyanonda P, Dagli C (1992) Composite stock cutting through simulated annealing. Mathemat Comput Model 16(1):57–74
25. Morabito R, Arenales M (2000) Optimizing the cutting of stock plates in a furniture company. Int J Prod Res 38(12):2725–2742
26. Morabito R, Belluzzo L (2007) Optimizing the cutting of wood fibre plates in the hardboard industry. Eur J Operat Res 183(3):1405–1420
27. Pierce F (1964) Some large scale production scheduling problems in the paper industry. Prentice-Hall, Englewood Cliffs
28. Prais M, Ribeiro C (2000) Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. INFORMS J Comp 12(3):164–176
29. Prasad SS, Dhande SG (1986) Computer aided design of optimal pattern layouts. CAD/CAM/CAE for Industrial Progress, pp 105–117
30. Qu W, Sanders J (1987) A nesting algorithm for irregular parts and factors affecting trim losses. Int J Prod Res 25(3):381–397
31. Roberts SA (1984) Application of heuristic techniques to the cutting-stock problem for worktops. J Oper Res Soc 35:369–377
32. Sarker BR (1988) An optimum solution for one-dimensional slitting problems: a dynamic programming approach. J Operat Res Soc 39(8):749–755
33. Toledo F, Carravilla M, Ribeiro C, Oliveira J, Gomes A (2013) The dotted-board model: a new MIP model for nesting irregular shapes. Int J Prod Econ 145(2):478–487
34. Verkhoturov M, Sergeyeva O (2000) The sequential value correction method for the two-dimensional irregular cutting stock problem. Pes Oper 20(2):233–246
35. Wascher G, Haubner H, Schumann H (2007) An improved typology of cutting and packing problems. Eur J Operat Res 183(3):1109–1130
36. Yaodong C, Yiping L (2009) Heuristic algorithm for a cutting stock problem in the steel bridge construction. Comp Operat Res 36(2):612–622