Unconstrained *k*-staged two-dimensional guillotineable single knapsack problem

Problema da mochila bidimensional irrestrito guilhotinado de *k*-etapas

*David Álvarez-Martínez**

*Luis Miguel Escobar Falcón***

*Ramón Alfonso Gallego-Rendón****

*Universidade Estadual Paulista ï Faculdade de Engenharia de Ilha Solteira ï São Paulo ï Brasil

**Joven Investigador COLCIENCIAS ï Universidad Tecnológica de Pereira ï Facultad de Ingenierías ï Programa de Ingeniería de Sistemas y Computación ï Risaralda ï Colombia

***Universidad Tecnológica de Pereira ï Facultad de Ingenierías ï Programa de Ingeniería Eléctrica ï Risaralda ï Colombia

ABSTRACT

The cutting and packing problems are considered classic problems in the operations research of high interest due to its big spectrum of application in the industry, and the development needed on the treatment of high mathematical and computational complexity structures. In this study is presented the unconstrained k-staged two-dimensional guillotineable single knapsack problem of rectangular items, with and without associated weights to the items, with and without 90 degrees rotations to the items, and with guillotine cuts. An appropriate encoding of the problem is presented to work on it using a metaheuristic hybrid algorithm of variable neighborhood search and simulated annealing. To check the efficiency of the presented methodology, cases of study were taken from specialized literature in order to analize and compare the presented solution method with the state-of-the-art of the problems. Results of excellent quality and never reported before on the literature were obtained in this work.

KEY WORDS. Unconstrained *k*-staged two-dimensional knapsack problem, variable neighborhood search, simulated annealing.

RESUMO

Os problemas de empacotamento e corte ótimo são considerados clássicos na pesquisa operacional devido a seu grande campo de aplicação na indústria e a sua alta complexidade matemática e computacional. Neste trabalho é apresentado o problema de empacotamento ótimo bidimensional irrestrito de peças retangulares numa placa só, com e sem pesos associados às peças, considerando a possibilidade de rotacionar 90° as peças, com restrições de corte tipo guilhotina e um número k máximo de etapas de corte (amplamente conhecido como o problema da mochila bidimensional irrestrita de *k*-etapas). É proposto um tipo de codificação para ser aplicada neste problema e resolver-lo mediante um algoritmo de otimização que combina as principais características do *variable neighborhood search* and *simulated annealing*. Para comprovar a eficiência do método apresentado foram usados casos de teste tomados da literatura especializada, onde são analisados e comparados os métodos de solução apresentados com aqueles do estado da arte do problema, obtendo resultados de excelente qualidade e nunca antes reportados na literatura.

PALAVRAS CHAVE. Mochila bidimensional irrestrita de *k*-etapas, variable neighborhood search, simulated annealing.

1. Introduction

Unconstrained *k*-staged two-dimensional guillotineable single knapsack problem is used to solve cutting problems when the material employed is one rectangular piece where the smaller rectangular pieces must be located, knowing the size and the associated cost for each of the small pieces, the objective is to maximize the value of the cutted pieces.

The characteristics of this problem are:

i) The associated cost can be related or not with the area of the piece that is going to be located; if the cost is equal to the area of the piece the problem will be solved without weights (unweighted version) and if the cost is different to the area of the item the problem will be solved with weights (weighted version).

ii) The orientation of the pieces that are going to be located with length *l* and width w is different from a piece of length *l* and width *w* (without rotation). If we consider that the dimensions (*l*,*w*) and (*w*,*l*) represent the dimensions of the same piece, we are talking a problem with rotation.

iii) The cutting patterns are guillotine type, each cut produces two sub-rectangles. The cuts are done from one edge to the opposite edge of the original rectangle.

iv) Exists a maximum number of stages *k*, taking the constant *k* < Ð as the sum of all the vertical and/or horizontal parallel cuts. If *k* is equal to 2, the problem is 2-staged (having this one a lot of real applications). If *k* is equal to 3, the problem is 3-staged. Finally, if a large value is assumed for *k*, represents a non-staged problem.

v) There is no maximum limit of the quantity of pieces that will be cut from each type (unconstrained version).

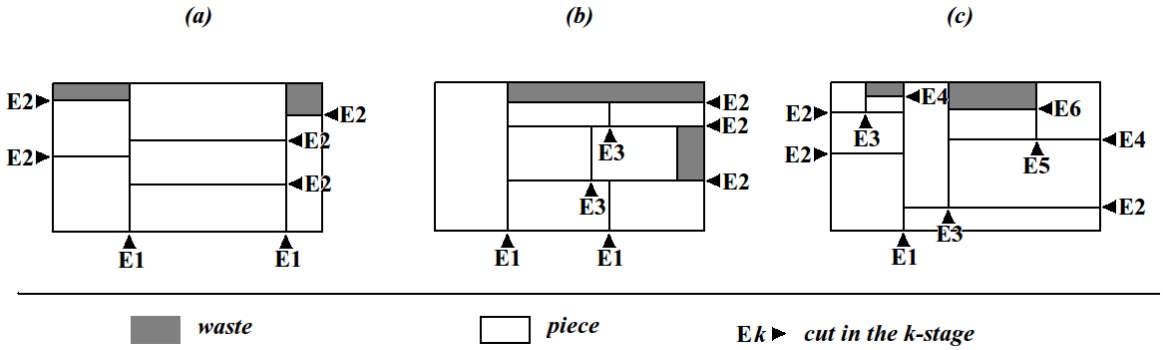

Figure 1: Representation of three solutions when *k* is fixed to 2, 3 and not-fixed respectively; Numbers by the cuts are the stage at which the cut is made. (a) a solution of the 2-staged problem: the value ó1ô corresponds to the first cut and ó2ô means that the second cut is applied; in this case they are 2 vertical cuts (counted one time) and 5 horizontal cuts (counted one time); (b) a solution to the 3-staged problem: The value ó1ô represents the first cut, ó2ô denotes the second cut and ó3ô is the third cut; in this case, they are: two vertical cuts (counted one time), 3 horizontal cuts (counted one time) and two internal vertical cuts (counted one time); (c) a solution to the non-staged problem.

(Gilmore and Gomoroy, 1965; 1966) proposed a recursive exact algorithm based on dynamic programming to solve the problem. Their algorithm is applicable to both weighted and unweighted versions. (Herz, 1972) propose a recursive tree search method, his method is more effective than Gilmore and Gomoryôs algorithm for an unweighted problem, but does not apply to weighted cases. (Beasley 1985) proposed an algorithm that is a modified version of Gilmore

and Gomoryôs algorithm. (Hifi and Zissimopoulos, 1996) proposed a recursive exact algorithm that uses a dynamic programming procedure and efficient lower and upper bounds. (G and Kang, 2002) improved Hiffi and Zissimopoulosô recursive algorithm by applying a more efficient upper bound. This is presently the most efficient exact algorithm for the unconstrained problem.

There are two general techniques used to solve constrained problems: top-down and bottom-up approaches. (Christofides and Whitlock, 1977) originally proposed the top-down. The bottom-up approach requires an enormous amount of memory, for this reason its implementation is not attractive. However, (G *et al.*, 2003) proposed an algorithm that starts from an initial solution of good quality and uses the bottom-up algorithm as strategy to generate the branches, decreasing the number of nodes to explore.

This article uses a binary tree encoding, called slicing tree structure, which decomposes the problem into smaller packing problems to solve them through two algorithms, the first one is an enumerative algorithm to find the types of cut (horizontal or vertical), the second one is an hybrid metaheuristic algorithm that combines the main characteristics of the variable neighborhood search (VNS) and simulated annealing (SA) to determine the optimal cut distances. In order to proof the efficiency and quality of the results obtained with the proposed methodology, study cases of the specialized literature were used. Solutions of excellent quality were obtained which have never been reported. The structure of this article is as follows: problemôs description, a general description of the methodology used to solve the unconstrained *k*-staged two-dimensional guillotineable single knapsack problem, analysis result and conclusions.

2. Problem Description

The unconstrained knapsack problem has two variants: unweighted version and weighted version. The first instance of the problem (*k*-staged weighted version with rotation) is defined as a finite set of *n* rectangular pieces of given dimensions (length l_i and width w_i), where $i = 1, 2, \acute{e}, n.$, which are cutted from a rectangular initial plate (knapsack) with lenght L and width W. With each piece is associated a profit c_i. One piece (l_i, w_i) is equal to a piece (w_i, l_i). The objective is to find a cutting pattern of the plate maximizing the profit function, see the equation 1.

$$max \quad \sum_{i=1}^{n} c_i \cdot z_i \tag{1}$$

Where z_i is a binary variable that indicates if the piece i would be cutted or not.

Subject to:

- The packed (cutted) pieces cannot trespass the limits of the plate.
- The pieces cannot overlap to each other.
- Only cuts of guillotine type are permitted.
- The cutting pattern should be obtained on k stages of cut.

The second instance of the problem (*k*-staged weighted version without rotation) only has one difference with the previous definition, and is the orientation of the pieces, therefore a piece (l_i, w_i) is not equal to a piece (w_i, l_i).

The third instance of the problem (*k*-staged unweighted version with rotation) only differs of the first instance on the associated profit c_i, because it is equal to l_i (length) multiplied

by w_i (width). Then, the objective consists on maximizing the area produced by the set of pieces and one piece (l_i, w_i) is equal to a piece (w_i, l_i).

The fourth instance of the problem (k-staged unweighted version without rotation) only differs from the previous definition on the orientation of the pieces, because a piece with dimensions (l_i, w_i) is not equal to a piece (w_i, l_i).

The same four previous instances are taken into account when it does not exist a maximum number of cutting stages (non-staged versions). In order to simplify, the notation presented by (Hifi, 2001) was used to enumerate the proposed problems on this study, where the most common k values are two and three.

- FUUkSK: represents the Fixed Unconstrained Unweighted k-Staged Single Knapsack problem;
- RUWkSK: denotes the Rotated Unconstrained Weighted k-Staged Single Knapsack problem;
- FUUnSK: corresponds to the Fixed Unconstrained Unweighted Non-Staged Single Knapsack problem;
- RUWnSK: denotes the Rotated Unconstrained Weighted Non-Staged Single Knapsack problem.

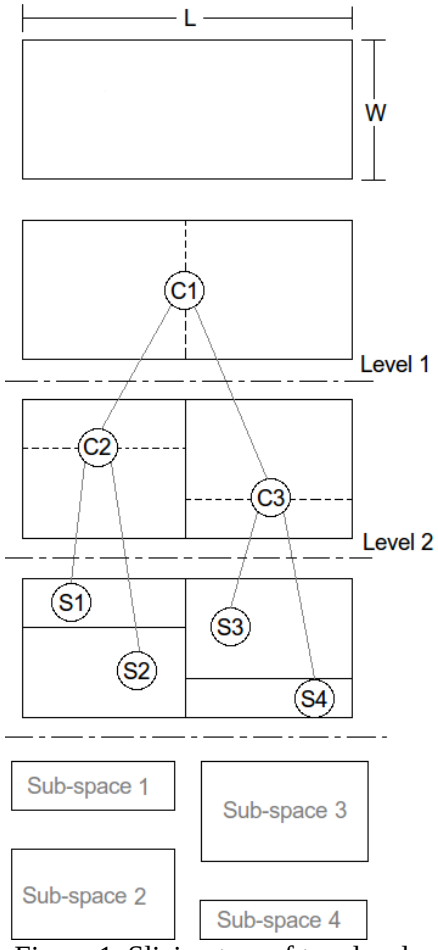

Figure 1. Slicing tree of two levels.

3. Methodology

Encoding

(Wong *et al.*, 1988) presented a data encoding for the floorplan design problem called slicing tree. A slicing tree is defined as: a tree with root, where each intern node (parent node) represents the position and how the cut will be done on the material (horizontal or vertical), meanwhile the terminal nodes (leaf nodes) represent the dimensions of the sub-spaces generated for cutting the grouped pieces.

One of the main advantages of using the slicing tree is the generation of cutting patterns guillotine type. Different proposed methodologies have corroborated the effectiveness of the slicing tree encoding, especially the ones presented by (Kroguer, 1995), (Cui, 2005 and 2007) and (Toro *et al.*, 2008).

Figure 1 illustrates a slicing tree for the problem, in that one the root node (Level 1) indicates where and how to make a perpendicular cut to the length of the knapsack. The hierarchy of the tree indicates that the left child node makes a perpendicular cut to the width of the left resulting subspace and the right child node makes a new perpendicular cut to the width of the right resulting subspace, this is for the second level. Then, on the third level to the left we have the subspaces 1 and 2 and to the right we have the subspaces 3 and 4. On each generated subspace are placed the biggest amount of pieces with the same shape (in order to keep the guillotine type constraints) maximizing the used area (or maximize the associated profit for the instances with this variant).

Therefore, the slicing tree contains on its internal nodes the information about the orientation of the cut (perpendicular to the length or to the width) and the distance where the cut should be done. By other hand its leaf nodes contain the dimensions of the resulting subspaces.

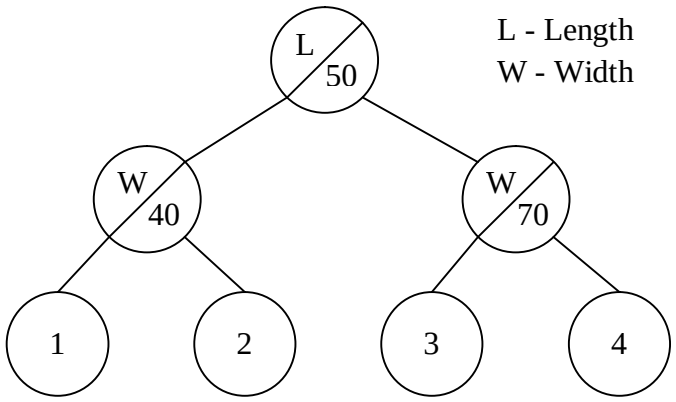

Figure 2. Example of slicing tree.

For instance, the Figure 2 presents the slicing tree of a knapsack with length and width 260 and 120 units respectively. The dimensions of the resulting subspaces will be presented on the Table 1. Note that each node of the slicing tree, except the leaves, contains the orientation and the distance of the cuts (percentage from zero to hundred). Hence, the root node indicates that a perpendicular cut to the length of the plate at the 50% of the knapsack will be done, dividing it into two new subspaces (note that the cut is guillotine type) both with equal dimensions, length 130 and width 120 units. So on, the tree will guide the cut process and the leaves will give us the dimensions of the subspaces.

Subspaces	Length	Width
1	130	48
2	130	72
3	130	84
4	130	36

Table 1. Dimensions of the subspaces.

Objective function calculation

After obtaining the sub-spaces, the placement of the pieces should be performed, this process is developed through a constructive best-fit algorithm, keeping the guillotine type restrictions and packing the biggest quantity of pieces for each sub-space.

The constructive best-fit algorithm consists on finding the set of identical pieces that maximizes the area of the subspace j (represented by its length and width, L_j and W_j respectively) with the best associated profit. The calculation of the objective function consists on applying the equation (2) to each sub-space.

$$\max\left\{\left(\left\lfloor\frac{L_j}{l_i}\right\rfloor\cdot\left\lfloor\frac{W_j}{w_i}\right\rfloor\right)\cdot c_i\right\};\forall i; i=1,2,...,n \qquad (2)$$

The calculation of the objective function consists on appling the best-fit algorithm to each sub-space, then the value of the objective function is the sum of the areas (or associated profits) packed on the knapsack.

The Table 2 presents a set of available pieces to be packed on the plate of the last example. Combining this information with the Table 1 we have enough data to calculate the objective function.

Code of the Piece	Length	Width	Associated Profit
1	20	15	300
2	30	25	750
3	15	15	225
4	7	25	175
5	40	25	1000

Table 2. Example of available pieces (note that the associated profit is equal to the area of each piece, thus this example is an unweighted problem)

The value of the objective function is equal to 25.650 square units, knowing the dimensions of the plate we can calculate the use percentage of the knapsack 25.650/32.000 = 80,16%. The placement of the pieces is presented on the Table 3.

Subspace	Code of the Piece	Quantity of Pieces	Profit
1	3	18	5.400
2	1	24	7.200
3	4	78	9.450
4	3	16	3.600
Total Profit			25.650

Table 3. Placement of the pieces for the example.

Different proposals that use the slicing tree codification look to find the optimal tree, having this process a difficult solution (Kroguer, 1995; Cui, 2005 y 2007), in contrast with (Toro

et al., 2008) that delimits and reduces the number of trees during the optimization. (Toro *et al.*, 2008) after making a statistical study, the slicing tree is defined with complete binary trees with three levels. (The Figure 3 shows a tree with three levels).

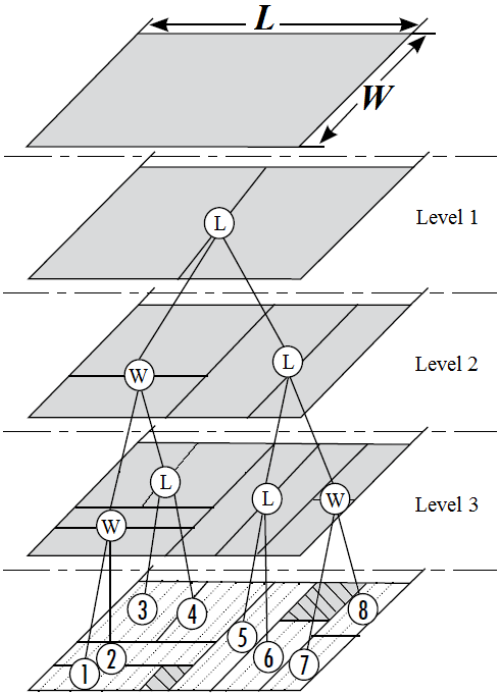

Figure 3. Representation through a slicing tree of three levels.

Optimization Scheme

The proposed codification in this study guarantees the feasibility of the guillotine type constraints and the maximum number of cutting stages. The tree of orientation cut and the tree with the cut distances are independent from each other (represented with the variables *O* and *D* respectively), this means that for each set of *O* values exists an optimal solution D^*.

The optimization scheme for the knapsack problems is illustrated in figure 4, first the direction of the cuts will be established (horizontal or vertical) and then the optimal distances for making these cuts are calculated. The Algorithm I makes an exhaustive search (enumerates all the possible solutions) over the *O* tree, whereas that the Algorithm II receives the *O* trees, and it must find the optimal distances for each one of them. The Algorithm *II* belongs to the metaheuristic techniques.

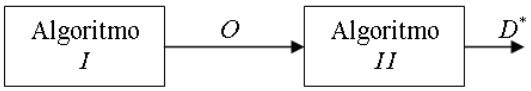

Figure 4. Optimization Scheme.

Algorithm *I*

Generates the possible trees of orientation cut through an enumerative technique. Due to the fact that the maximum number of depth level of the tree is fixed, the result of the algorithm will be: two orientation trees for the first level, eight trees for the second level, and 128 trees for the third level, each one of them is used as an input for the Algorithm II.

Algorithm *II*

This algorithm uses metaheuristic optimization techniques, combining the main features of the variable neighborhood search (VNS; Mladenovic and Hansen, 1997) and simulated annealing (SA; Kirkpatrick *et al.* 1983). The first one is considered the main algorithm, the second one was used as a specialized transition method to make local searches and to produce changes to the nodes in the tree of distances, using the strategy of the SA (temperature variation).

The variable neighborhood search consists on making local searches over a set of defined neighborhoods, i.e., based on a solution a new neighborhood is picked to look for a better solution. In case of finding it, this one will be the new starting point, in the opposite case a new neighborhood must be generated in order to make a new inspection.

The set of neighborhoods N_i (where, i={1,2,..., i_{max}}; and i_{max} is the number of father nodes of the slicing tree) are defined as the number of nodes that must change its value.

$i=1$, then, N_1 = {one random number must change its value}

$i=2$, then, N_2 = {two random numbers must change their values}

$i=k$, then, N_k = {k random numbers must change their values}

Therefore, the resultant set of neighborhoods is $N = \{N_1, N_2,..., N_k,..., N_{max}\}$.

The transition mechanism consists in permitting big changes on the distances of the cuts during the first iterations, like the simulated annealing accepts the lost of quality for the objective function at the beginning of the process. As the iterations advance, transitions become more deterministic. The transition is defined as the modification of a nodeôs value from the distances tree, trough the mechanism shown in the equation (3).

$$node\ i = node\ i + \left(rand - \frac{1}{2} \right) \cdot \left(\sqrt{1 - \frac{k}{TotalIterations}} + \varepsilon \right) \qquad (3)$$

Equation (3) is composed by: the current value of the node i from the distances tree, the number of the current iteration, the number of total iterations and $\mathring{U}$ is the minimum percentage to generate a change on the distances, where $\mathring{U}$= 100/max(L,W).

The proposed algorithm is a trajectory-based metaheuristic algorithm, and it works as follows:

1. Start from a random solution D.
2. Define a neighborhood for D, generate a solution D_T into the neighborhood through the transition mechanism previously defined.
3. Evaluate if the objective function of the solution D_T is better than the one produced by D, if this is true D is replaced by D_T and the process continues defining a new neighborhood for this solution, else a new and bigger neighborhood is calculated and a new D_T is generated.

Figure 5 illustrates the flowchart of the algorithm A_{VNS+SA}. The given name to this algorithm is because it uses a transition mechanism that emulates the temperature variable from the simulated annealing.

Step 1	Initialize *Total_Iterations*, *Number_Of_Levels* Let *N* = SetOfNeighborhoods(*Number_Of_Levels*) Let *i* = 1 Let *k* = 1 Let *D* = GenerateInitialRandomSolution(*Number_Of_Levels*) Go to Step 2
Step 2	Let *Incumbent* = *D* Go to Step 3
Step 3	Equation (3) ï Let D_T = TransitionMechanism(*D*, *N*, *k*, *Total_Iterations*) Let *k* = *k* + 1 Go to Step 4
Step 3.2	If CalculateObjectiveFunction(D_T) > CalculateObjectiveFunction (*D*) Go to Step 3.2.1. Else Go to Step 3.2.2.
Step 3.2.1.	Let *D* = D_T Let *i* = 1 If CalculateObjectiveFunction (*D*) > CalculateObjectiveFunction (*Incumbent*) Go to Step 3.2.3. Else Go to Step 4
Step 3.2.2.	Let *Incumbent* = *D* Go to Step 4
Step 4	Let *i* = *i* + 1 Go to Step 5
Step 5	If *k* = *Total_Iterations* Terminate algorithm Else Go to Step 2

Figure 5. Steps of the Hybrid Algorithm of Variable Neighborhood Search and Simulated Annealing

Solution Methodology

Therefore, the proposed methodology on this study is a search descending, where the trees *O* and *D* can be treated together or individually depending on the process stage. A scheme of the methodology is presented on the Figure 6. The steps that must be done are described and depending of the instance of the problem some steps are omitted.

Moreover, the type of searches that are presented on this work are only different on the number of levels of the slicing tree. This permits relate these levels with the cutting constraints by stages, i.e., for the 2-staged problems of the methodology is executed until step 2, for the 3-staged problems until step 3 and finally for the non-staged problems is executed the whole methodology.

Calibration of parameters

The parameter adjustment is very important in order to have good results with the metaheuristic techniques. Different approaches were presented to make the parametrization. In general, there are no exact and efficient methods to make the parameter adjustment of the different metaheuristic techniques, commonly these algorithms are parameterized through the combination of an exhaustive search and a statistical analysis to the quality of the results.

Step 1.

Search over a tree of level 1.

Over the tree *O* (orientation of the cuts) an exhaustive search is done. Meanwhile over the *D* tree (distance of the cuts) the hybrid algorithm of variable neighborhood search and simulated annealing is used.

Step 2.

Search over a tree of level 2.

Over the tree *O* (orientation of the cuts) an exhaustive search is done. Meanwhile over the *D* tree (distance of the cuts) the hybrid algorithm of variable neighborhood search and simulated annealing is used.

Step 3.

Search over a tree of level 3.

The trees *O* and *D* are treated as a unique tree that will be optimized by the hybrid algorithm of variable neighborhood search and simulated annealing.

Step 4. Deepening.

Searches over the trees of level 2.

The eight resulting subspaces from the optimal tree in the step 3, are sent to the step 2 in order to make an improvement process to the solution obtained.

Figure 6. Scheme of the solution methodology.

To make the parameterization, different studies suggest: classify the test problems (if they exist) by its complexity (mathematical or computational), choose one representing problem (candidate) from each class, make an adjustment to the parameters for each candidate trough an exhaustive mesh search and finally recombine the obtained parameters for each class picking the best combination.

This process requires a great computational effort because the parameter adjustment through a mesh search represents another optimization process of almost the same complexity to the problem of this study because each parameter belongs to a range of values.

In this case the strongest constraints for the parameter adjustments are the rotation of the pieces and the maximum number of cutting stages. In the Table 4 the values of the parameters are shown.

Parameters	**Problem**	
	Without Rotations	**With Rotations**
Total Iterations Level 1	100	200
Total Iterations Level 2	500	1000
Total Iterations Level 3	1000	2000
Total Iterations Deepening	2000	4000

Table 4. Parameters and values for each type of problem.

4. Results Analysis

The test systems used in this study were taken from the specialized literature; both approximate and exact methodologies are used in the solution of the mentioned problems. The selected problems are diverse in terms of the mathematical complexity and were specially designed for each type of problem.

Thirty test cases were selected for the two-dimensional guillotineable knapsack problem, 15 cases for the weighted version ([UW1-UW11] and [UWL1-UWL4]) and 15 cases for the unweighted version ([UU1-UU11] and [UUL1-UUL4]). These 30 cases present different types of knapsack with distributions between 25 and 200 pieces, the data base is presented by (Hifi, 2001) and its available online (Hifi, 1997). Of all cases, 22 ([UW1-UW11] and [UU1-UU11]) belong to the category of packing problems with middle mathematical complexity and 8 ([UWL1-UWL4] y [UUL1-UUL4]) belong to a category of high mathematical complexity. Different studies have used these test cases to prove the performance of the proposed methodologies.

All the algorithms were developed on Delphi 7.0 ® and executed on a computer with a Pentium ® IV processor of 3,0 GHz and a RAM memory of 1 GB.

	2-staged			
	With Rotation			
	Weighted		**Unweighted**	
	Quality	**Time**	**Quality**	**Time**
Reached	15		15	
Improved		**4**		**4**
Below		11		11

Table 5. Comparison between the A_{VNS+SA} with the best known solutions.

	3-staged							
	Weighted				**Unweighted**			
	Without Rotation		**With Rotation**		**Without Rotation**		**With Rotation**	
	Quality	**Time**	**Quality**	**Time**	**Quality**	**Time**	**Quality**	**Time**
Reached	11		11		7		5	
Improved	**4**	**5**	**4**	**15**	**4**	**6**	**4**	**15**
Below		10			4	9	6	

Table 6. Comparison between the A_{VNS+SA} with the best known solutions.

Once the paper is published, these problems will be made available on the Internet at http://utp.edu.co/~planeamiento/dinop/librerias/knapsack/index.html. The best reported solution (Best Known Solution, BKS) in the specialized literature is the one presented on the web site for the test cases of each type of problem along with the employed computational time. For the 2-staged knapsack problems without piece rotation and the non-staged knapsack with and without piece rotation are not published. On the other hand, for the results of the remaining problems (Hifi, 1998) has the best solutions, but also (G *et al.*, 2003) achieves good results. Cases like UWL1, UWL2, UWL3, UWL4, UUL1, UUL2, UUL3 and UUL4 for the 3-staged knapsack problem without piece rotation do not have an answer reported because most of the approximations to these cases of high mathematical complexity were done through exact

techniques where the computational effort is too high for its solution. Therefore, some authors omit reporting them.

The Tables 5 and 6 show a comparative summary of the achieved answers by the presented methodology and the best reported solutions on the specialized literature. For the 2-staged knapsack problem with pieces rotation and the non-staged with and without pieces rotation, the comparison is not done due that do not exist references to validate the results.

The tables 5 and 6 show that the A_{VNS+SA} algorithm presented good quality results for the different types of problems, demonstrating that the proposed methodology is robust for the different variants of the unconstrained knapsack. It is notable that the obtained results for the non-staged problems are the same optimal values of the 3-staged problems, this validates the proposal of limiting the slicing tree to three levels, aiming to reduce the search space with a low risk of losing good quality solutions.

5. Conclusions

The unconstrained two-dimensional guillotineable problem has been solved with all its variants: weighted, unweighted, with rotation, without rotation, 2-staged, 3-staged and non-staged. Using a hybrid algorithm of variable neighborhood search and simulated annealing, the results achieved were of good quality.

The slicing tree encoding from the floorplan design problem was adapted in this work, combining a tree with binary values for the orientation slicing and another tree with real values to determine the distance of the cuts. This encoding proposal presented a satisfactory performance for this type of problems because it reduces the search space without the risk of losing good quality solutions.

An optimization algorithm that combines the main features of variable neighborhood search and simulated annealing was implemented. The first one is considered the main algorithm, the second one was used as a specialized transition method to make local searches and it changes the nodes on the tree of distances. At the beginning of the process random values were used and at the end the values were mainly deterministic.

The computational times obtained using the proposed methodology, in some cases were better than the ones reported on the specialized literature but due to the differences between the processors architecture and the programming languages used, is not possible to make a final conclusion for the methodologies. In general we can say that the times were reasonable.

The solution method used presented a satisfactory performance on the final results with the different variants of the worked problem, this allows us to conclude that the methodology is robust and can be used on similar problems like the constrained knapsack, bin packing, strip packing and others. This can also be used for three-dimensional packing problems.

References

BEASLEY, J. E. Algorithms for unconstrained two-dimensional guillotine cutting. *J. Oper. Res. Soc.*, vol. 36, pp. 297ï 306, 1985.

BEN, S.; CHU, C. y ESPINOUSE, M. L. Characterization and modelling of guillotine constraints. *European Journal of Operational Research*, vol. 191, pp. 112ï 126, 2008.

CHRISTOFIDES, N. y WHITLOCK, C. An algorithm for two-dimensional cutting problems. *Operations Research*, vol. 25, pp. 30-44, 1977.

CUI, Y. An exact algorithm for generating homogenous T-shape cutting patterns. *Computers & Operations Research*, vol. 34, pp. 1107-1120, 2007.

GALLEGO, R.; ESCOBAR, A. H. and TORO, E. M. *Técnicas metaheurísticas de optimización*, Textos universitarios, Universidad Tecnológica de Pereira, 2008.

G, Y.-G. y KANG, M.-K. A new upper bound for unconstrained two-dimensional cutting and packing. *J. Oper. Res. Soc.*, vol. 53, pp. 587ï 591, 2002.

G, Y.-G.; KANG, M.-K. and SEONG, J. A best-first branch and bound algorithm for unconstrained two-dimensional cutting problems. *Operations Research Letters*, vol. 31, pp. 301ï 307, 2003.

GILMORE, P. C.and GOMORY, R. E., Multistage cutting problems of two and more dimensions. *Operations Research*, vol. 13, pp. 94-120, 1965.

ð . The theory and computation of knapsack functions. *Oper. Res.*, vol. 14, pp. 1045-1074, 1966.

HERZ, J. C. A recursive computing procedure for two-dimensional stock cutting. *I.B.M. Ji Res. Dev.*, vol. 16, pp. 462-469, 1972.

HIFI, M. and ZISSIMOPOULOS, V. A recursive exact algorithm for weighted two-dimensional cutting. *European J. Oper. Res.*, vol. 91, pp. 553ï 564, 1996.

HIFI, M. *Problem instances for the 2D Cutting/Packing Problems*, [on line], 1997. <ftp://cermsem.univ-paris1.fr/pub/CERMSEM/hifi/2Dcutting/>.

ð . Exact algorithms for the guillotine strip cutting/packing problem. *Computers & Operations Research*, vol. 25, pp. 925ï 940, 1998.

ð , M. Exact algorithms for large-scale unconstrained two and three staged cutting problems. *Computational Optimization and Applications*, vol. 18, pp. 63 ï 88, 2001.

KIRKPATRICK S., GELATT C., and VECCHI M., (1983). ñOptimization by simulated annealingò, *Science*, Vol. 220, pp. 671-680.

KRÖGER B., (1995). ñGuillotineable bin packing: A genetic approachò, *European Journal of Operational Research*, Vol. 84, pp. 645-661.

MLADENOVIC N. and HANSEN P., (1997). ñVariable neighborhood searchò, *Computers and Operations Research*, Vol. 24, pp. 1097ï 1100.

TORO, E.; GARCÉS, A. and RUIZ, H. Solución al problema de empaquetamiento bidimensional usando un algoritmo híbrido constructivo de búsqueda en vecindad variable y recocido simulado. *Revista Fac. de Ing. Universidad de Antioquia*, vol. 46, pp. 119-131, 2008.

WONG, D. F.; LEONG, H. W. and LIU, C. L. *Simulated Annealing for VLSI Design*. Kluwer Academic Publishers, 1988.