Journal of Computing and Information Technology - CIT 22, 2014, 4, 251–265
doi:10.2498/cit.1002422

A Parallel Hyper-heuristic Approach for the Two-dimensional Rectangular Strip-packing Problem

Istvan Borgulya

Faculty of Business and Economics, University of Pecs, Hungary

In this paper, we present a parallel hyper-heuristic approach for two-dimensional rectangular strip-packing problems (2DSP). This is an island model with a special master-slave structure, and in all the islands we run a memetic algorithm-based hyper-heuristic (HH). The basic technique of this HH is a memory-based evolutionary technique, the "extended virtual loser" (EVL). The memory-based technique memorises the past events, e.g., past successes of the evolutionary process or bad values of the variables; thus, we can influence the operations of the evolutionary algorithms using this memory. The EVL technique learns the bad values of the variables based on the worst solutions of the population and computes probabilities to control the mutation steps. With the help of the EVL technique, we can use a mutation-omitting recombination operator and obtain a learning mechanism for the selection of heuristics. In the HH, the selection of the low-level heuristics is modified with mutations based on the EVL technique using a local search. The island model achieved good performance. The test instances show that the proposed algorithm is efficient for the rectangular strip-packing problem.

Keywords: rectangular strip-packing, hyper-heuristic, memetic algorithm, memory-based technique, island model

1. Introduction

In the 2DSP, we have a set of n rectangular items with w_i widths and h_i heights, where $i = 1, 2, \ldots, n$. The goal is to pack all the items without overlap into a strip of width W such that the height of the packing is minimised. The packing problem can be specified with orientation and guillotine constraints. We will regard the 2DSP variant with a fixed orientation and without a guillotine cut.

The 2DSP belongs to cutting and packing problems that have several industrial applications, such as garment manufacturing, sheet metal cutting, furniture making, and shoe manufacturing, and also applications in logistics. The 2DSP is NP-hard (Garey and Johnson 1979). Many exact, heuristic and meta-heuristic algorithms have been published to solve it. Usually, a combination of different heuristics can give better solutions than a single heuristics. It is a complex task to determine the best algorithm including the operators and heuristics to use. In this planning process the HHs can help. A HH is a search method that works in the heuristic space (using only limited information from the problem solution space) selecting or generating good heuristics and then applying them to solve the problem. In the last decade, HH methods were also used for the solution of the 2DSP.

In this paper, we are interested in HH methods and we developed a HH for the 2DSP. Our motivation was to build a HH algorithm using a special learning technique and to show the application of this HH. The learning technique of this HH is a memory-based evolutionary technique, the EVL technique (Borgulya 2006). The memory-based technique memorises the past events, e.g., past successes of the evolutionary process or bad values of the variables; thus, we can influence operations of the evolutionary algorithms using this memory. The EVL technique learns the bad values of the variables based on the worst solutions of the population and computes the probabilities to control the mutation steps. If we use the EVL technique in the EA, we can use mutations that omit the recombination operator and obtain a learning

mechanism for the selection of heuristics in a HH (Borgulya 2006, 2008).

For the HH we use a memetic algorithm (MA); Mas are hybrid evolutionary algorithms (EA) incorporating a local search (LS) process. In this MA-based HH, we apply a mixture of three placement heuristics which are modifications of earlier placement heuristics and use placement strategies and improving procedures that are different from the earlier ones. The HH uses a set of mutation heuristics and a set of LSs for the solution. Selection of the mutation, LS and placement heuristics are modified with the mutations based on the EVL technique using a local search. For better results, we also use parallel computing. We apply an island model (Borgulya 2010), and in all the islands, we run the MA-based HH. To the best of our knowledge, this model is the first to present parallel HH for the 2DSP.

Thus, our contribution is a new parallel HH algorithm for the 2DSP. The key features of this contribution are the following:

- We propose a parallel MA-based HH for 2DSP. This is an island model, and the HH uses EVL-based mutations and local search to control the application of the heuristics and select other heuristics.
- We propose three placement heuristics which are modifications of the BF and BL heuristics. All new heuristics use placement strategies and improving procedures that are different from the earlier ones.
- Our algorithm is efficient for the rectangular strip-packing problem.

The rest of this paper is organised as follows. Section 2 gives a summary of the methods of the 2DSP and the methods of HHs. Section 3 gives a description of the island model used. Section 4 presents the new MA-based hyper-heuristic for the 2DSP, gives a description of the EVL technique, as well as of the operations and characteristics of the algorithm; and it also gives examples for the individuals and operators. Section 5 presents the low-level heuristics used. The computational results are reported in Section 6, and the conclusions are summarised in Section 7.

2. Related Works

2.1. Exact, Heuristic and Meta-heuristic Methods for 2DSP

We find exact, heuristic and meta-heuristic algorithms to solve the 2DSP. The first exact algorithm was a linear programming approach (Gilmore and Gomory 1961). Tree search-based algorithms were published for the guillotine and non-guillotine versions (Christofides and Whitlock 1997). Some authors used variants of the branch and bound technique (e.g., Martello et al. 2003; Arahori et al. 2013; Boschetti and Monatelli 2010). We can use these methods to solve only small 2DSP instances within a reasonable time.

Heuristic algorithms can find the solutions quickly, but do not guarantee to provide the global optimum. The most important heuristic group, constructive heuristics, use various strategies for the packing of the items. An important point is how they choose the next item for placement. They can choose the next item from a fixed sequence of the items or, dynamically, from the sequence for placement. A fixed ordered sequence is used for the best known placement heuristic: the "bottom up, left justified" (BL) heuristic (Baker et al. 1980). BL first sorts the items according to their areas and then starts with each item from the top-right corner. Then, it slides the item as far as possible to the lowest location and then, as far as possible to the left of the strip. There are more improved versions of BL (e.g., Hopper and Turton 2001). The "best fit" (BF) heuristic (Burke et al. 2004) dynamically chooses the items. BF repeats two operations until all items are placed; it searches for an available space as low as possible and then places the item that fits the space best. An improved version of BF is the "bidirectional best-fit" heuristic (Özcan et al. 2013). Additional heuristics are the squeaky wheel optimisation (Burke et al. 2011), the shacking procedure (Wauters et al. 2013) and a combination of constructive heuristics with other methods, e.g., BF and simulated annealing (Burke et al. 2004).

We can use the usual meta-heuristic methods for the 2DSP. Meta-heuristics usually give sequences of the items for other heuristics to use. The most frequently used meta-heuristics are

simulated annealing (SA), the tabu search (TS) and the genetic algorithm (GA) (e.g., Jakobs 1996; Faina 1999; Burke et al. 2011; Thomas and Chaudhari 2014). The methods with best results are usually hybrid methods that can use combinations of meta-heuristics, LSs, placement heuristics or improving procedures (e.g., Jakobs 1996; Yang et al. 2013; Wei et al. 2011; Iori et al. 2003). Some heuristics use other solutions for the improvement. The reactive GRASP (Alvarez-Valdes et al. 2008) involves a constructive phase and a subsequent iterative improvement phase.

2.2. Hyper-heuristic Methods

The HH is "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems" (Burke et al. 2013). The HH works in the heuristic space selecting or generating good heuristics and applies them to solve the problem. It uses only limited information from the problem solution space. If the HH selects heuristics, it selects one or more heuristics from a low-level heuristic set in every step of the HH algorithm. The low-level heuristics are usually mutation operators, local search procedures, and other constructive heuristics. If the HH generates heuristics, it uses genetic programming (GP) to build new heuristics.

The HH selects or generates heuristics; in both cases, we can use constructive and improvement techniques. The constructive techniques construct the solution step-by-step, with low-level heuristics; the improvement techniques change only small parts of the solution. Selection methods differ between the two techniques. Constructive techniques use various methods to select appropriate heuristics, e.g., EAs, TSs, and classification systems. The selection mechanism in the improvement techniques is divided into two parts: a heuristic selection method and a move acceptance criterion. The heuristic selection method can select heuristics randomly or it can select heuristics using a learning mechanism. The move acceptance criterion, which is based on the result, accepts or rejects the new solution.

Most HHs work with one solution (single-point search). However, there are a few HHs that work with multiple solutions (population). These are generally improvement techniques and are built based on GA, ACO, or PSO, for example, (detailed descriptions of the HH methods are available in Burke et al. 2013).

In the following, we describe three HHs for the 2DSP:

- Garrido and Riff (2007) developed a constructive, GA-based HH. Individuals of a population are sequences of low-level heuristics. The low-level heuristic set is four placement heuristics, and the individuals are sequences of these low-level heuristics with ordering, rotation and repeating information. In a generation, it uses roulette wheel selection, one-point crossover and three mutation operators to generate descendants. It constructs multiple solutions in every generation and it accepts the descendants (the descendants constitute a new population).
- The HH of Burke et al. (2010) is a GA with an improvement technique and works with a population of the solutions together with the information about the heuristics. The low-level heuristic set includes three placement heuristics with four placement strategies. The individuals are permutations of items; a set of candidate heuristics, together with probabilistic information, is attached to each item of the individual. For every item, the algorithm uses roulette-wheel selection to choose a heuristic from the candidate set in the packing process. The GA applies crossover and mutation operations on the permutation of items to generate descendants, and a learning mechanism updates the probabilities of applying heuristics and/or modifies the candidate heuristic set according to the result of the descendant. The algorithm accepts the descendants (the descendants constitute a new population).
- The HH of Nguyen et al. (2012) is a GP-based HH that automatically generates programs as new placement heuristics. It is similar to GP in Burke et al. (2010). The individual is a program that can use variables which store important data on the placement and influence the result (e.g., the slot height and width, the width and height of the item, the difference between the slot and item widths). To improve the results, the generated heuristics also include the statistics

from previous packing solutions, which allow the evolved heuristics to iteratively correct the mistakes made in previous placement decisions. Based on this statistics, the GP calculates the average penalty of the item, which indicates the difficulty of placing. In each placement step, the heuristic will calculate the score for each combination of items, allocation and slot (together with the penalty), and the combination with the highest score will be applied for the next placement. The algorithm accepts the new heuristics (they constitute a new population).

2.3. Parallel Methods

Parallel processing is a useful tool for reducing runtimes and improving the quality of meta-heuristics. In most applications of parallel processing, the parallel meta-heuristics are implemented using a master–slave or an island model. For cutting and packing problems, there are also parallel solutions. For instance, for bin packing, circles packing, and 2D cutting problems, we find parallel processing beneficial. There are also a few parallel HHs for timetabling and scheduling (Rattadilok et al. 2005) and for real parameter optimisation (Biazzini et al. 2009).

3. The Island Model

The island model (Borgulya 2010) uses a master-slave structure with a centralised scheme in which slave processors execute the evolutionary process and periodically send their best partial results to a master process. The master process stores partial results in a common migration set (*MS*) and then randomly chooses individuals from the *MS*, one after another for every slave and sends them to the slaves.

The island model can work with *np* parallel processors (e.g., 2, 4, 8 or 16), and the given EA runs on every parallel processor. The parallel process will be controlled with the *frequ* and *mignumb* parameters. Let EP_i be the evolutionary process on the i*th* slave processor with P_i population. *frequ* determines the communication frequency; after *frequ* iterations, every slave sends migrating individuals into the *MS*. Every EP_i sends the *mignumb* number of individuals to the *MS*, and these select the best individuals for migration. The master process randomly selects *mignumb* individuals from *MS* for every EP_i, and each population P_i obtains these individuals from *MS* (The master process selects the individuals for P_i randomly, except for the earlier migrant individuals from P_i). Every EP_i replaces the worst individuals with the incoming ones in P_i.

We simulated the island model in one processor and did not examine the parallel environment characteristics in a network. To determine the cost, we computed only the running time, which also includes the communication time. Naturally, we considered all parallel processes as only one process time, which belonged to the longest process.

In our parallel HH approach (PMAHH), the MA-based HH (MAHH) runs as an evolutionary process in this island model.

4. Our Hyper-heuristic Algorithm

4.1. The EVL Technique

Our MAHH uses the EVL technique in different mutation operations. Mutation operators of MAHH select another low-level mutation heuristic, another low-level local search heuristic and other placement heuristics in a few positions of the placement heuristics list. Among the low-level mutation heuristics for the solution, there are also two heuristics that use mutation based on the EVL (see Sections 4.2 and 5.3).

The principle of the EVL is as follows (Borgulya 2006). Let us consider a generic EA, and suppose that the individual has l variables, each having k discrete values. Notice that the ECM (explicit collective memory) is a $k \times l$ matrix that stores and learns the relative frequencies of different values of the variables. This matrix is updated throughout the evolution procedure, using some of the worst performing individuals.

Let ECM_{ij}^{gen} be the collected relative frequency of the *i*th values on the *j*th position (variable) until the *gen*th generation. We can update the elements of the ECM matrix

$$ECM_{ij}^{gen+1} = (1-\alpha)\, ECM_{ij}^{gen} + \alpha \Delta ECM_{ij}$$

where ΔECM_{ij} is a relative frequency of the ith value on the position j based on the worse individuals of the genth generation and α denotes some relaxation factor (e.g., $\alpha = 0.2$). We update ECM periodically, every knth generation (e.g., $kn = 10$). The computation of ΔECM_{ij} goes on as follows:

- we take 20% of the worst individuals from the population;
- we count how many times the ith value occurs on position j in the worst individuals $(i = 1, 2, \ldots, k; j = 1, 2, \ldots, l)$
- we divide the ΔECM matrix by the number of worst individuals.

We use the ECM matrix to estimate the probability of the mutation. Let X be an individual, and let z be the value of the jth variable in individual X $(X_j = z)$. To determine the probability of mutating the jth variable in individual X, we use the formula

$$pr_j = 1 - \left| \frac{ECM_{zj}^{gen}}{\sum_{i=1}^{k} ECM_{ij}^{gen}} - a_j \right|$$

where a_j is the following: if B is one of the best individuals, then *if* $z = B_j$ *then* $a_j = 1$ *else* $a_j = 0$.

4.2. The Memetic Algorithm-based Hyper-heuristic

Our MAHH uses an improvement technique and works with a population. The individual has two parts: a permutation of n items and a set of the low-level heuristics used. We acquired a new solution by applying these low-level heuristics from an earlier individual. We define mutation operations and LS for the heuristics part of the individual. Mutation is based on the EVL technique; thus, in the algorithm, there is no separated learning mechanism.

MAHH is a steady-state MA-based HH. It uses a two-stage algorithm structure to speed up convergence and to produce higher-quality results. The first stage is a quick "preparatory" stage that is designated to improve the quality of the initial population. The second stage is a steady-state MA that searches for better solutions.

For certain tasks, the algorithm might "get stuck" at one of the local optima. To enable escape toward a potential global optimum, the algorithm generates new, additional individuals. A new individual is also a descendant and can help to improve capability and speed of the algorithm to find the global optimum. Thus, in the second stage, new descendants are periodically inserted in the population until the maximum size of the population is reached.

Algorithm 1 shows the main steps of MAHH and Figure 1 shows how it works. Parameters of the algorithm are the following:

tmax – the maximum size of the population.

t – the size of the population in the first stage.

itt – the number of generations in the first stage

kn – the algorithm is controlled in every *kn*th generation.

timeend – the limit of the running time.

tp – parameter of truncation selection.

gp, *rp* – parameters of the condition of the *Restart* procedure.

LSn, LSmax, rep, prun1, prun2 – parameters of the low-level local search heuristics.

Hn – parameter of the LSH local search.

Imax – parameter of the placement heuristics.

The operations and the characteristics are as follows:

Input. The algorithm reads the instance and the values of the parameters (these are described and given in the parameter selection section).

Individuals. Every individual of population P is a pair of the permutation of the n items and a set of heuristics $I_j = (S_j, H_j)$ $j = 1, 2, \ldots, |P|$, where S_j is the permutation of the items and H_j is the list of the low-level heuristics used. In the list, we distinguish three groups: mutation heuristics, local search heuristics and placement heuristics. The H_j list of heuristics is (MS_j, LS_j, PL_{j1}, PL_{j2},. . . ,PL_{jn}), where MS_j is a mutation heuristic on the permutation, LS_j is a local search heuristic on the permutation and the PL_{j1}, $PL_{j2}, \ldots, PL_{jn}$ heuristics are the placement heuristics; there is one placement heuristic attached to every position of the permutation (see Section 5).

Initial population. In the initial population, the algorithm generates five individuals where

Algorithm 1. The main steps of MAHH

```
Input: the instance, the values of the parameters.
Initialize the individuals,the ECMs matrices.
/* First stage */
Do itt times
    Generate a descendant randomly.
    Accept descendant according to the move acceptance criteria.
    In every knth generation update ECMs. Filter.
od.
/* Second stage */
Repeat
    Do kntimes
        Select a parent.
        Mutations of the low level heuristics.
        Apply the low level heuristics.
        LS on the low level heuristics.
        Accept descendant according to the move acceptance criteria.
    od
    If (t< tmax) then    t=t+1 fi
    Apply LS procedures on the last descendant.
    Accept descendant according to the move acceptance criteria.
    Apply LS procedures on the best individual.
    Accept individual according to the move acceptance criteria.
    Update ECMs. Filter, Restart.
until running time>timeend
end
```

items of the permutation are ordered using various criteria (decreasing order: by height, height and width, perimeter, area and by width and height) and the H_j sets are generated randomly. Other individuals are generated randomly.

Fitness function. The fitness function is defined on the permutation and it is the height of the packed items on the strip.

Selection operator. MAHH selects individual I_j based on truncation selection. In this selection, only the best *tp* percentage of the population is considered a potential parent.

Mutation operators. MAHH applies appropriate mutations for the three groups of the heuristics in H_j. All mutations use the EVL technique with different ECMMS, ECMLS and ECMPL matrices:

- Mutation of MS_j. The ECMMS is a 4×1 column matrix; every mutation heuristic has a row in the matrix. The mutation operator selects another heuristic with the highest pr_j probability. The algorithm applies the mutations with 0.5 probabilities.
- Mutation of LS_j. The ECMLS is a 7×1 column matrix; every local search heuristic has a row in ECMLS. The mutation operator selects another local search with the highest pr_j probability. The algorithm applies the mutations with 0.5 probabilities.
- Mutation of the list of the placement heuristics. The ECMPL is a $3 \times n$ matrix. Every placement heuristic has a row, and every position on the list has a column in ECMPL. The mutation operator first chooses a random i position on the list of the placement heuristics. Next, it chooses another heuristic for this position with the highest pr_i probability. (MAHH applies this mutation operator twice in a sequential manner). The algorithm applies the mutations with 0.5 probabilities.

Local search. For the 2DSP, we use three different placement heuristics, named HP1, HP2 and HP3. For the list of placement heuristics in the individuals, we define a local search, named LSH. LSH is a complex local search. In the first step, it constructs new sequences of the placement heuristics. If the application of a se-

quence improves the result, it will be the new sequence of the placement heuristics in the individual. The generated sequences are the following: with *ph* probability, LSH takes the HP1 on every position; otherwise, the HP2 or HP3 are taken with equal probabilities. Nine sequences are generated with the following *ph* values: 0.1, 0.2. . . 0.9. Construction of the nine sequences is repeated five times. In the second step, LSH modifies the sequence of the placement heuristics. At a random position, it changes the heuristic randomly to a different one. If the application of the sequence improves the result, the modified sequence will be the new sequence of the placement heuristics in the individual. This modification is repeated *Hn* times.

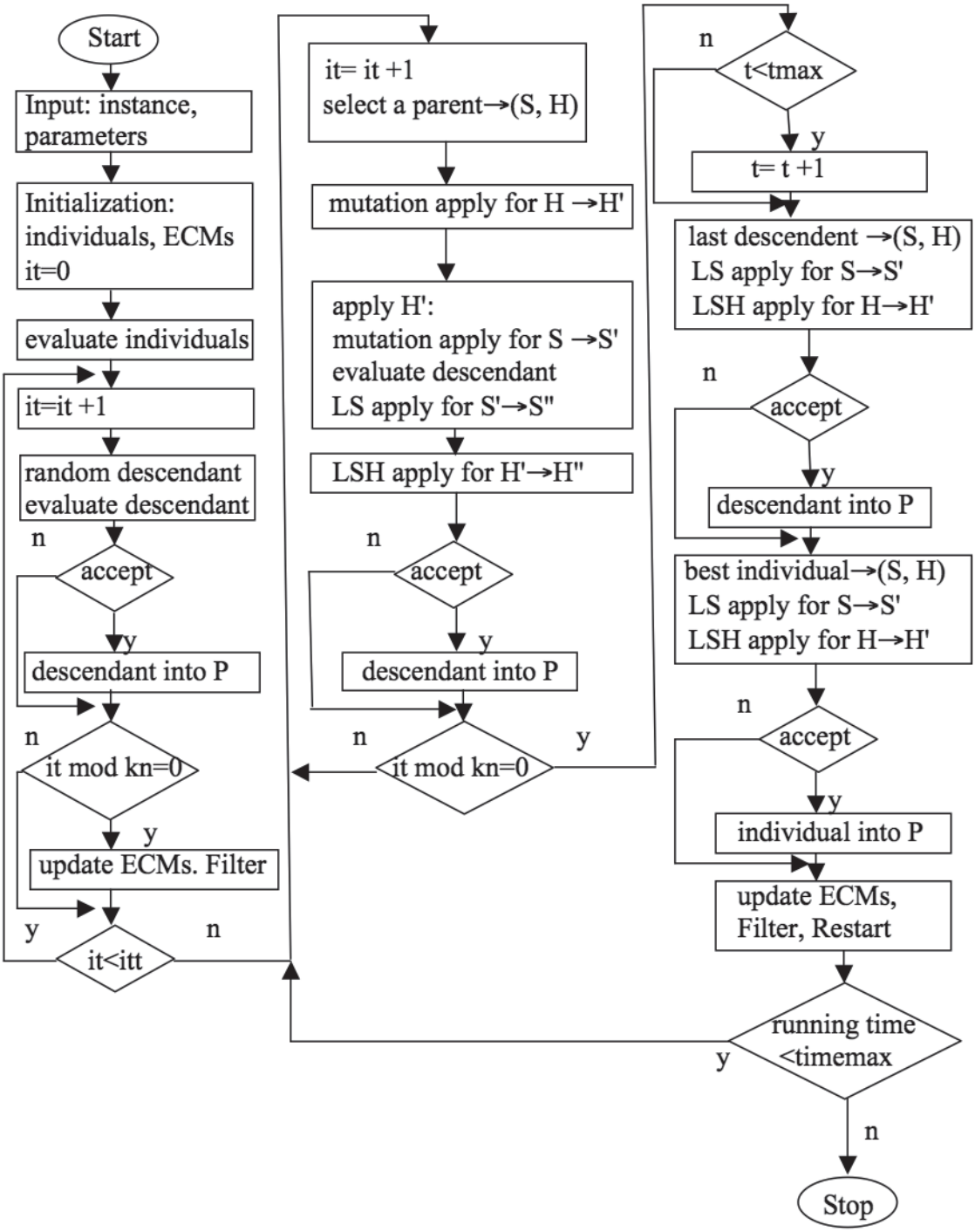

Figure 1. The flowchart of MAHH.

Filter, Restart. To speed up the convergence, the algorithm uses *Filter* and *Restart* procedures. The *Filter* procedure filters and deletes the weak individuals which are members of the neighbourhood of a better individual (individuals I_k and I_j are neighbours if the Hamming distance $d^H(S_k, S_j) < 3$). If the fittest solution did not change in the last *gp* generations, the *Restart* procedure deletes the weakest solutions (*rp* proportion of the population).

Move acceptance criterion. The move acceptance criterion is a special variant of the crowding technique and it helps separate the local and global optima. This crowding technique compares the descendant with the former solutions (or the parent) based on similarities in the S_j permutations. In the first stage, the descendant may replace (is accepted) the most similar of the former solutions if the descendant is better (similarity is based on the Hamming distance); in the second stage, the descendant may replace (is accepted) the parent if the descendant is better. If the descendant is an additional individual or if there are fewer individuals than the size of the population (after *Restart* or *Filter* procedures), a new descendant is unconditionally inserted (is accepted) into the population until the population size is reached.

Stopping criterion. The algorithm is terminated if the running time limit is reached.

4.3. Example

The following example demonstrates the individuals and how MAHH works. Let there be 5 items ($n = 5$). Among the low-level heuristics we distinguish three groups: mutation heuristics, local search heuristics and placement heuristics groups. Let's take four mutation heuristics, seven local search procedures and three placement heuristics. We identify the items and the low-level heuristics in every group with sets of serial numbers. The individual is composed of 4 parts: permutation of items, serial number of the mutation, serial number of the LS and a sequence of serial numbers of the placement heuristics. Permutation and sequence of the placement heuristics jointly constitute the solution.

Let's take a random individual (1, 3, 5, 2, 4, 1, 3, 1, 2, 1, 3, 3), the 1, 3, 5, 2, 4 is the permutation, and from the low-level heuristics we have the 1st mutation heuristic, the 3rd local search heuristic and the sequence 1st, 2nd, 1st, 3rd, 3rd of the placement heuristics.

Let us see the applications of the mutations of MAHH, of the low-level heuristics and of the local search procedure of MAHH.

a) Let us suppose that the mutations of MAHH arrived to the following modifications of the serial numbers: the 3rd mutation, the 1st local search and on the 2nd and 3rd positions of the sequence the new serial numbers are 1st and 2nd. The descendant is thus the following: (1, 3, 5, 2, 4, **3**, **1**, 1, **1**, **2**, 3, 3) (modifications are shown in bold).

b) Let's now demonstrate application of the low-level heuristics: *Application of the 3rd mutation*: let us suppose the mutation is a random insert operation and the item from the 5th position will be inserted to the 2nd position. The descendant is thus the following: (1, **4, 3, 5, 2**, 3, 1, 1, 1, 2, 3, 3).
Application of the 1st local search: let us suppose that it analyses the swaps in the permutation. After every swap it computes the new value of the fitness and if the result is better, it accepts the swap. Let us suppose the two swaps improved the fitness: between the 2nd-4th and 3rd-5th positions. The new descendant is thus the following: (1, **5**, **2**, **4**, **3**, 3, 1, 1, 1, 2, 3, 3).
Application of the placement heuristics. This is the computation of the fitness. One after the other, the placement heuristics place the next not-yet-placed item, or a dynamically selected not-yet-placed item from the permutation part of the individual.

c) LSH is a local search procedure of MAHH. It modifies multiple times the sequence of placement heuristics. For example, the outcome may be the following after the application of the LSH on the last descendant: (1, 5, 2, 4, 3, 3, 1, **2, 2, 2**, **1**, 3). LSH computes a new value of the fitness and if the result is better, it accepts the modification.

5. Low-level Heuristics for the 2DSP in MAHH

Among the low-level heuristics, we distinguish three groups: placement heuristics, local search heuristics and mutation heuristics groups.

5.1. The Placement Heuristics

For the 2DSP, we use three different placement heuristics. The first is a modified version of the BL heuristic; the second and third ones are modified versions of the BF heuristic. The most important modification of the heuristics is that they use a local search procedure after every item, and the procedure attempts to improve the placement with a block of multiple items. We call these heuristics HP1, HP2 and HP3.

HP1 first applies the BL heuristic and then attempts to improve the result with a local search procedure. The BL places the next not-yet-placed item from the solution part of the individual. If the width of the packed item is smaller than the width of the space, HP1 applies the local search *ImpLS*, which works in two steps. First, it searches other items to be packed into the empty part of the space. These items are the next not-yet-packed items with maximum *imax*. In the second step, *ImpLS* builds blocks combining one, two or three items from the selected items and chooses the block that fits in the empty part of the space and has the largest width. If *ImpLS* finds a block, HP1 places the block into the space. (In the block, the items can be different, unlike in the reactive GRASP method Alvarez-Valdes et al. 2008).

HP2 first applies the BF heuristic and then attempts to improve the result in two different ways with the *ImpLS* local search. The BF dynamically selects a not-yet-placed item and fits the item into the lowest available space. If the lowest available space is too narrow to place a remaining item into it, the space is raised to the level of the lower space adjacent to it, and the two spaces are merged. Next, BF begins again. If the placing was successful and the width of the packed item is smaller than the width of the space, HP2 applies the *ImpLS* local search. First, HP2 applies the *ImpLS* for the full space. If *ImpLS* builds a block with a larger width than the fitted item found, the block replaces the item in the space. If there is no appropriate block, HP2 (similarly to HP1) applies the *ImpLS* on the empty part of the space. If *ImpLS* finds a block, HP2 places the block into the space.

HP3 is a modified version of HP2, in which the BF is replaced with a modified, score-based version of BF. The idea of the score-based BF was motivated by the IDBS method (Wei et al. 2011). In the IDBS, the placing heuristic uses several strategies, and one of the strategies assigns fitness values to some important (space, fit item) pairs. The IDBS may choose the (space, fit item) pair with the highest fitness value to place. In the score-based BF, we assign scores to some special types of spaces without the items. The score reflects the importance of the spaces, and our BF variant searches for a fit item for the space that has the highest score value. Thus, the score is 4 if the space is at the left or the right edge and lies deeper than the neighbouring space; 3 if the space lies deeper than its neighbours and the neighbouring spaces lie at the same height; 2 if the space lies deeper than its neighbours and the neighbours lie at different heights; 1 if the space is located in middle of three steps; and 0 if the space is located at the lowest available space.

It is important to determine the efficiency of the HP1, HP2 and HP3 heuristics. First we analysed the importance of the *ImpLS* local search and used the heuristics without *ImpLS* as well. Our conclusion is that the *ImpLS* can improve the results in general and gives better than average results. Next we compared the heuristics separately and some combinations of the

Used placement heuristics	C-6-100-1		C7_1		Zdf9	
	Best	Aver.	Best	Aver.	Best	Aver.
HP1	793	801.4	246	247.6	5975	6082.3
HP2	776	780.4	243	244.1	5667	5769.0
HP3	775	780.3	**242**	**243.1**	5566	5621.3
HP1, HP2	774	779.3	243	244.1	5625	5652.6
HP1, HP3	777	**778.4**	244	244.4	**5276**	**5357.3**
HP2, HP3	**771**	779.7	243	244.2	5517	5626.1
HP1, HP2, HP3	777	779.0	244	244.5	5480	5554.0

Table 1. Computational results with different placement heuristics.

heuristics in MAHH. Table 1 shows a comparison with the help of the C_6_100_1, C7_1 and Zdf9 instances (the best results for every instances are shown in bold). Among the HP1, HP2, and HP3 heuristics, the HP3 has the best results. The combination of two or three heuristics gives better result in general than the simple heuristics. Although the results of HP1 are weaker than the results of HP2 and HP3, their combination with HP1 improves the quality of the results. Thus, in the MAHH algorithm, we use the three (HP1, HP2, and HP3) heuristics together.

5.2. The Local Search Heuristics

To compile the set of appropriate LS procedures for the algorithm, we use seven LS procedures. A local search version takes every pair of the values of the permutation, attempts to apply a move on the pair and computes the new value of the fitness function. If the application of the move improves the result, the procedure accepts the move. The moves in the procedures are swap, insert, or inversion.

The low-level heuristics are the following: LS1 is a search with a swap move; LS2 is a search with an insert move; LS3 is a search with an inversion move; LS4 is LS1 followed by LS2; LS5 is LS1 followed by LS3; LS6 is LS2 followed by LS3; and LS7 is LS1 followed by LS3 and then by LS4. We attempted to achieve shorter running times with all local searches. Thus, every local search uses only a few random parts of the permutation for the move. The number of parts is controlled by the *LSn* parameter, and the length of each part is at most *LSmax* elements. If a local search improves the results, it can be repeated. The *rep* parameter controls the repetition of local searches. If *rep=1*, then MAHH can repeat the local search. There are two parameters *prun1* and *prun2* that control the applications of the local searches in the algorithm. If *prun1=1*, then MAHH can apply the local searches on every descendant. If *prun2=1*, the algorithm can apply the local searches on the last descendant in every *kn*th generation.

5.3. The Mutation Heuristics

The algorithm uses one of four mutation heuristics for the S_j permutations. The low-level mutation heuristics are the following: a swap based on the EVL technique, an insert based on the EVL technique, a random swap, and a random insert move.

The first and second mutation heuristics use a common ECM matrix, named ECMS. The ECMS is an $n \times n$ matrix. Every item has a row, and every position of the S_j permutation has a column in the matrix. If the mutation is based on the EVL method, the mutation heuristic first chooses a random k position in the permutation and then chooses another item for this position with the highest pr_k probability based on the ECMS; it searches the position of the new item in S_j and swaps the values of the positions or inserts the new item in position k. MAHH examines randomly only min $(n, 50)$ rows of the ECMS matrix to search for the highest pr_k probability.

6. Computation Experiments

The PMAHH algorithm was implemented in C++. It was executed on an iMAC with an Intel Core i5 2.5 GHz processor with 4 GB of RAM, running the Mac OS X 10.9.2 operating system.

We tested our algorithm with the benchmark instances that are used generally in publications. The zero-waste instances comprise the first group, and their optimal solutions are known. The second group includes non-zero-waste instances, the optimal solutions of which involve some wasted regions.

The zero-waste instances are the following:

- C (Hopper and Turton 2001) with 21 instances, the number of input rectangles ranging from 16 to 197.
- N (Burke et al. 2004) with 13 instances, the number of input rectangles ranging from 10 to 3152. We also used the set (N_{12}), without the N13 instance.
- RB (Ramesh Babu and Ramesh Babu 1999) with 1 instance, the number of input rectangles is 50.

- NT (Hopper 2000) with 70 instances, the number of input rectangles ranging from 17 to 199.
- CX (Pinto and Oliveira 2005) with 7 instances, the number of input rectangles ranging from 50 to 15 000.

The non-zero-waste instances are:

- NP (Nice1,. . . , Path5) (Wang and Valenzela 2001) with 12 instances, the number of input rectangles ranging from 25 to 1000.
- NPT (Nice1t,. . . , Path5t) (Wang and Valenzela 2001) with 60 instances, the number of input rectangles ranging from 1000 to 5000.
- ZDF (Leung et al. 2011). We use only the first 13 instances (ZDF_{13}), the number of input rectangles ranging from 580 to 15 096.
- 2sp (cgcut1, . . ., gcut1, . . ., ngcut1, . . ., beng1, . . ., beng10) (Beasley 1985, 1985a; Christofides and Hadjiconstantinou 1995; Bengtsson 1982) with 38 instances, the number of input rectangles ranging from 7 to 200.
- BWMV (Berkey and Wang 1987; Martello et al. 2003) with 500 instances, the number of input rectangles ranging from 20 to 100.

6.1. Parameter Selection

While studying some of the more complex problems of the benchmark sets, we analysed the process of MAHH to determine how the parameter values affect the convergence, the discovery of the global optimum and the speed of the calculation. Thus, we analysed the population size (t and $tmax$ parameter), the frequency of checks (kn parameter), the generation in the first stage (itt parameter), the parameters of the *Restart* procedures (gp and rp) and of the truncation selection (tp). Summarising the results of the analysis, we found the following: $t=5$, $tmax=30$, $itt=5$, $kn=5$, $gp=300$, $rp=0.7$ and $tp=0.1$, and we accepted that the published methods allowed a duration of 60 CPU seconds for each test problem.

In the second part of the parameter selection, we analysed the LS procedures, which are the most time consuming part of the algorithm. As the number of dimensions increases, their running times increase rapidly. Thus, the values of *LSp*, *LSmax*, *Hn, imax, rep* and the parameters *prun1* and *prun2* are important. We searched the parameter values at different values of n. The result is the following:

- $20 \leq n \leq 100$, we use the planned LS with repetition. The parameter values: *LSp=30, LSmax=50, Hn=30, imax=100, rep=1, prun1=1* and *prun2=1*.
- $100 < n \leq 200$, we narrow the use of the LSs for the last descendant and the best individual with repetition. The parameter values: *LSp=30, LSmax=50, Hn=30, imax=100, rep=1, prun1=0* and *prun2=1*.
- $200 < n < 500$, the above values of the parameters except: *LSp=3*.
- $500 \leq n < 2000$, the above values of the parameters, but the local searches run without repetition (*rep=0*).
- $2000 \leq n \leq 4000$, we use only the local search for the best individual without repetition. *LSp=2, LSmax=50, Hn=30, imax=100, rep=0, prun1=0* and *prun2=0*.
- $4000 < n \leq 8000$, we use only the local search for the best individual without repetition. *LSp=2, LSmax=10, Hn=10, imax=50, rep=0, prun1=0* and *prun2=0*.
- $8000 < n \leq 15100$, we use only the local search for the best individual without repetition. *LSp=1, LSmax=5, Hn=0, imax=10, rep=0, prun1=0* and *prun2=0*.

Based on these results, we can give the initial values of the parameters of the LSs. For example, if $n=300$, the values of the LS parameters are the following: *LSp=3, LSmax=50, Hn=30, imax=100, rep=1, prun1=0* and *prun2=1*.

In the third part of the parameter selection, we used the parameters of MAHH in the island model and searched appropriate parameter values for the island model. Experimenting with setting *frequ* and *mignumb* we found that using *frequ* $=kn$ and *mignumb* $=1$ yielded the best-quality results in general. We also applied the island model with different numbers of islands. We used 2, 4, 8, 16 and 32 islands and found the best-quality results using the 32-island version. Thus, we used the model with *np=32* and allowed a duration of 60 CPU seconds for each test problem.

6.2. Comparative Results

PMAHH was run 10 times on each test instance, and we provide the best and the average results for every instance as % gap, which is the percentage gap to the lower bound (LB), namely, % gap = 100 *(obtained solution - LB)/LB. LB is the optimal height for the zero-waste problem. For the non-zero-waste problem, LB is as described by Leung et al. (2011).

We show the summary of our results compared with some published results. The first comparison shows the best result of two placement heuristics, namely the BF (Burke et al. 2004) and BBFM (Özcan et al. 2013) placement heuristics, and the results of the following five HHs: the PMAHH, the GA of Garrido and Riff (2007) (R_GA), the GA of Burke et al. (2010) (B_GA), the GP of Burke et al. (2010a) (B_GP) and the GP of Nguyen et al. (2012) (N_GP). The second comparison is based on the best-published results in the papers of Wei et al. (2011) and Yang et al. (2013), where the running times were 60 CPU seconds for each test problem. From these papers, we chose four methods: the reactive GRASP (Alvarez-Valdes et al. 2008), the ISA (Leung et al. 2011), the SRA (Yang et al. 2013) and the IDBS (Wei et al. 2011).

In Tables 2-4, we show the comparisons. In Table 2, we find the average best results (% gap) of the placement heuristics and HHs. The results were available only for the C, N, N_{12}, RB and NP test sets. For C, the result of PMAHH is approximately twofold better than the second best result of R_GA. For N and N_{12}, the result of PMAHH is approximately twofold better, and for NP, the result is approximately 75% better than the second best results of N_GP. We can conclude that for the C, N, N_{12} and NP test sets, our approach outperforms the BF and BBFM placement heuristics and the four HHs.

Table 3 shows the average results, and Table 4 shows the best results of GRASP, ISA, SRA, PMAHH and IDBS. For the zero-waste test sets (C, N, RB, NT and CX), the IDBS is the best method based on the best and the average results. For the RB test set, only the GRASP did not find the optimal result.

	GRASP	**ISA**	**SRA**	**IDBS**	**PMAHH**
C	0.95	0.76	0.69	**0.12**	0.81
N	0.95	0.41	0.23	**0**	0.40
RB	0.30	**0**	**0**	**0**	**0**
NT	2.32	2.24	1.60	**1.54**	2.23
CX	0.88	0.88	0.52	**0.43**	0.50
NP	3.06	2.52	**2.00**	2.10	2.62
NPT	1.50	0.56	**0.15**	0.35	0.89
ZDF_{13}	-	4.00	2.94	-	**2.50**
2sp	2.68	3.02	3.07	3.01	**2.63**
BWMV	1.77	1.70	**1.63**	2.00	1.70

Table 3. Average results (% gap) of the methods (with 60 CPU seconds).

	GRASP	**ISA**	**SRA**	**IDBS**	**PMAHH**
C	0.95	0.64	0.62	**0.04**	0.67
N	0.95	0.20	0.13	**0**	0.23
RB	0.30	**0**	**0**	**0**	**0**
NT	2.32	1.96	1.30	**1.01**	1.69
CX	0.88	0.67	0.45	**0.40**	**0.40**
NP	3.06	2.32	**1.90**	**1.90**	2.37
NPT	1.50	0.56	**0.15**	0.25	0.78
ZDF_{13}	-	3.08	2.87	-	**1.80**
2sp	2.68	2.99	3.05	2.62	**2.60**
BWMV	1.77	1.53	**1.49**	1.76	1.53

Table 4. Average best results (% gap) of the methods (with 60 CPU seconds).

	BF	**BBFM**	**R_GA**	**B_GA**	**B_GP**	**N_GP** (60 sec)	**PMAHH** (60 sec)
C	5.70	2.32	1.39	-	10.50	1.70	**0.67**
N_{12}	4.32	1.48	-	4.76	3.50	0.58	**0.24**
N	4.05	1.37	-	-	-	0.53	**0.23**
RB	6.66	**0**	-	-	-	-	**0**
NP	6.43	5.18	-	-	-	3.19	**2.37**

Table 2. Average best results (% gap) of placement heuristics and HHs.

For the CX test set, PMAHH is also the best method based on the best results and the second of the five methods based on the average results. Based on the best and average results of the NT test sets, PMAHH ranks third of the five methods, and based on its results for the C and N test sets, it is the fourth of the five methods. However, for the C test set, the best results of ISA, SRA and PMAHH are very similar and for the C, N and NT test sets, the average results of PMAHH and ISA are very similar.

For the non-zero-waste test sets, there is not a best method for every test set. For the NP, NPT and BWMV test sets, the SRA is the best method based on the best and average results, and for the ZDF_{13} and 2sp test sets, the PMAHH is the best method based on the best and the average results. For the BWMV sets, the PMAHH and ISA are the second best methods, and for the NP and NPT test sets, the PMAHH is ranked fourth based on the best and the average results (in the three tables, the best results for every test set are shown in bold).

We can conclude that our algorithm is efficient for the rectangular strip-packing problem; PMAHH is the best methods (or equally good) in case of four test sets and the second best method in case of the largest BWMV test set. Based on the test results, it is better than reactive GRASP, and PMAHH belongs to the group of ISA, SRA and IDBS.

7. Conclusions

In this paper, we present a parallel memetic algorithm-based hyper-heuristic for the 2DSP. In our algorithm, the individuals are (solution, low-level heuristics) pairs. The algorithm organises the choice of the low-level heuristics with an improving technique that uses the memory-based EVL technique. The algorithm uses three placement heuristics that are modified versions of the BL and BF placement heuristics and uses improving local searches when placing an item. Our algorithm is efficient for the rectangular strip-packing problem; based on the test results, it belongs to the group of ISA, SRA and IDBS methods. For four test sets, it exhibited the best results.

In the future, we plan to extend our algorithm to solve other types of strip-packing problems. Our plan is to develop a hyper-heuristic framework based on the memetic algorithm-based hyper-heuristic.

References

[1] R. Alvarez-valdes, F. Parreño, J. M. Tamarit, Reactive GRASP for the strip packing problem. *Computers & Operations Research* **35**, 1065–1083, 2008.

[2] Y. Arahori, T. Imamichi, H. Nagamochi, An exact strip packing algorithm based on canonical forms. *Computers & Operations Research*, **39**(12), 2991–3011, 2013.

[3] B. S. Baker, E. G. Coffman, R. L. Rivest, Orthogonal packing in two dimensions. *SIAM Journal on Computing* **9**, 846–55, 1980.

[4] J. E. Beasley, Algorithms for unconstrained two-dimensional guillotine cutting. *The Journal of the Operational Research Society* **36**, 297–306, 1985.

[5] J. E. Beasley, An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research* **33**, 49–64, 1985.

[6] B. E. Bengtsson, Packing rectangular pieces – a heuristic approach. *The Computer Journal* **25**, 253–7, 1982.

[7] J. O. Berkey, P. Y. Wang, Two-dimensional finite bin-packing algorithms. *The Journal of the Operational Research Society* **38**, 423–429, 1987.

[8] M. Biazzini, B. Banhelyi, A. Montresor, M. Jelasity, Distributed hyper-heuristics for real parameter optimization. *Genetic and Evolutionary Computation Conference*, (F. Rothlauf, ed.). ACM: New York, NY, 1339–1346, 2009.

[9] I. Borgulya, An Evolutionary Algorithm for the Biobjective QAP. Computational Intelligence, Theory and Applications "Advances in Soft Computing" (B. Reusch, ed.), Springer series, 577–586, 2006.

[10] I. Borgulya, An Algorithm for the Capacitated Vehicle Routing Problem with Route Balancing. *Central European Journal of Operations Research*, **16**(4), 331–344, 2008.

[11] I. Borgulya, An Island Model for the No-Wait Flow Shop Scheduling Problem. *PPSN XI*. Krakow, Springer, LNCS 6239, 280–289, 2010.

[12] M. A. Boschetti, L. Monatelli, An Exact Algorithm for the Two-Dimensional Strip Packing Problem. *Operations research* **58**, 1774–1791, 2010.

[13] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, R. Qu, Hyper-heuristics: a survey of the state of art. *Journal of the Operational Research Society*, 1–30, 2013.

[14] E. K. BURKE, Q. GUO, G. KENDALL, A Hyper-Heuristic Approach to Strip Packing Problems. In *PPSN XI*. Krakow, LNCS 6238, 465–474, 2010.

[15] E. K. BURKE, M. HYDE, G. KENDALL, A Genetic Programming Hyper-Heuristic Approach for Evolving 2-D Strip Packing Heuristics. *IEEE Transactions on Evolutionary Computation*, **14**(6), 942–958, 2010.

[16] E. K. BURKE, G. KENDALL, G. WHITWELL, A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research* **52**, 655–71, 2004.

[17] E. K. BURKE, M. HYDE, G. KENDALL, A Squeaky Wheel Optimization Methodology for Two Dimensional Strip Packing. *Computers & Operations Research*, **38**(7), 1035–1044, 2011.

[18] N. CHRISTOFIDES, E. HADJICONSTANTINOU, An exact algorithm for orthogonal 2-D cutting problems using guillotine cuts. *European Journal of Operational Research* **83**, 21–38, 1995.

[19] N. CHRISTOFIDES, C. WHITLOCK, An algorithm for two-dimensional cutting problems. *Operations Research* **25**, 30–44, 1997.

[20] L. FAINA, An application of simulated annealing to the cutting stock problem. *European Journal of Operational Research*, **114**(3), 542–556, 1999.

[21] M. R. GAREY, D. S. JOHNSON, Computers and intractability: a guide to the theory of NP-completeness. New York: Freeman, 1979.

[22] P. GARRIDO, M. C. RIFF, An Evolutionary Hyperheuristic to Solve Strip-Packing Problems H. IDEAL. (YIN ET AL. ED.), LNCS 4881, 406–415, 2007.

[23] P. GILMORE, R. GOMORY, A linear programming approach to the cutting stock problem. *Operations Research* **9**, 849–859, 1961.

[24] E. HOPPER, Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods. Ph.D. thesis, Cardi University, 2000.

[25] E. HOPPER, C. H. TURTON, An empirical investigation of metaheuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research* **128**, 34–57, 2001.

[26] M. IORI, S. MARTELLO, M. MONACI, Metaheuristic algorithms for the strip packing problem.: Optimization and Industry: New Frontiers. (P. PARDALOS, V. KOROTKICH, ED.), Kluwer Pub. 159–179, 2003.

[27] S. JAKOBS, On genetic algorithms for the packing of polygons. *European Journal of Operational Research 88*, 165–181, 1996.

[28] S. C. H. LEUNG, D. ZHANG, K. M. SIM, A two-stage intelligent search algorithm for the two-dimensional strip packing problem. *European Journal of Operational Research*, **215**(1), 57–69, 2011.

[29] S. MARTELLO, M. MONACI, D. VIGO, An exact approach to the strip packing problem. *INFORMS Journal on Computing*, **15**(3), 310–319, 2003.

[30] S. NGUYEN, M. ZHANG, M. JOHNSON, K. C. TAN, Automatic Discovery of Optimization Search Heuristics for Two Dimensional Strip Packing Using Genetic Programming. SEAL (L. T. BUI ET AL. EDS.), LNCS 7673, 341–350, 2012.

[31] E. ÖZCAN, Z. KAI, H. DRAKE, Bidirectional best-fit heuristic considering compound placement for two dimensional orthogonal rectangular strip packing. *Expert Systems with Applications* **40**, 4035–4043, 2013.

[32] E. PINTO, J. F. OLIVEIRA, Algorithm based on graphs for the non-guillotinable two-dimensional packing problem. *Second ESICUP Meeting*, Southampton, 2005.

[33] A. RAMESH BABU, N. RAMESH BABU, Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. *International Journal of Production Research*, **37**(7), 1625–43, 1999.

[34] P. RATTADILOK, A. GAW, R. S. K. KWAN, Distributed choice function hyper-heuristics for timetabling and scheduling. The Practice and Theory of Automated Timetabling V: Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling (E. K. BURKE, M. TRICK, EDS), LNCS 3616, Springer: Berlin, 51–70, 2005.

[35] J. THOMAS, N. S. CHAUDHARI, A new metaheuristic genetic-based placement algorithm for 2D strip packing. *J Ind Eng Int*, 10–47, 2014.

[36] C. L. VALENZELA, P. Y. WANG, Heuristics for large strip packing problems with guillotine patterns: an empirical study. *Proceedings of the 4th Metaheuristics International Conference*. University of Porto, Porto, Portugal, 417–421, 2001.

[37] P. Y. WANG, C. L. VALENZELA, Data set generation for rectangular placement problems. *European Journal of Operational Research* **134**, 378–391, 2001.

[38] T. WAUTERS, J. VERSTICHEL, V. BERGHE, An effective shaking procedure for 2D and 3D strip packing problems. *Computers & Operations Research* **40**, 2662–2669, 2013.

[39] L. WEI, W. C. OON, W. ZHU, A. LIM, A skyline heuristic for the 2D rectangular packing and strip packing problems. *European Journal of Operational Research*, **215**(2), 337–46, 2011.

[40] S. YANG, S. HAN, W. YE, A simple randomized algorithm for two-dimensional strip packing. *Computer & Operations Research*, **40**(1), 1–8, 2013.

Received: May, 2014
Revised: September, 2014
Accepted: October, 2014

Contact address:
István Borgulya
University of Pécs
Faculty of Business and Economics
Rákóczi út 80.
H-7621 Pécs
Hungary
e-mail: borgulya@ktk.pte.hu

ISTVÁN BORGULYA received his diploma in applied mathematics from the University of Szeged, Hungary in 1971. He received his PhD degree in computer science and the habilitation in economics from the University of Pécs too. He has been working 40 years at the University of Pécs, Hungary. He teaches informatics for economics and informatics students. He was head of the Department of Business Informatics and head of the BSc Business Informatics Study. Recently he became a honorary professor. His research interests include fuzzy methods in the decision support and optimization with evolutionary algorithms.