Int. J. Production Economics 145 (2013) 547–560

Contents lists available at ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

A genetic algorithm for two-dimensional bin packing with due dates

Julia A. Bennell [a,*], Lai Soon Lee [b], Chris N. Potts [c]

[a] School of Management, University of Southampton, Southampton SO17 1BJ, UK
[b] Department of Mathematics, Universiti Putra Malaysia, 43400 Serdang, Selangor, Malaysia
[c] School of Mathematics, University of Southampton, Southampton SO17 1BJ, UK

ARTICLE INFO

Article history:
Received 15 February 2012
Accepted 21 April 2013
Available online 2 May 2013

Keywords:
Cutting and packing
Two-dimensional bin packing
Due date
Scheduling
Genetic algorithms

ABSTRACT

This paper considers a new variant of the two-dimensional bin packing problem where each rectangle is assigned a due date and each bin has a fixed processing time. Hence the objective is not only to minimize the number of bins, but also to minimize the maximum lateness of the rectangles. This problem is motivated by the cutting of stock sheets and the potential increased efficiency that might be gained by drawing on a larger pool of demand pieces by mixing orders, while also aiming to ensure a certain level of customer service. We propose a genetic algorithm for searching the solution space, which uses a new placement heuristic for decoding the gene based on the best fit heuristic designed for the strip packing problems. The genetic algorithm employs an innovative crossover operator that considers several different children from each pair of parents. Further, the dual objective is optimized hierarchically with the primary objective periodically alternating between maximum lateness and number of bins. As a result, the approach produces several non-dominated solutions with different trade-offs. Two further approaches are implemented. One is based on a previous Unified Tabu Search, suitably modified to tackle this revised problem. The other is randomized descent and serves as a benchmark for comparing the results. Comprehensive computational results are presented, which show that the Unified Tabu Search still works well in minimizing the bins, but the genetic algorithm performs slightly better. When also considering maximum lateness, the genetic algorithm is considerably better.

1. Introduction

Cutting and packing problems have been the subject of extensive research over a number of years motivated by a wide range of real world applications. A typology of the cutting and packing literature can be found in Wäscher et al. (2007). This paper focusses on two-dimensional packing in which rectangles of specified dimensions are to be cut from identical stock sheets. It is related to the "two-dimensional, rectangular single bin size, bin packing problem", which is known as 2DRSBSBPP in Wäscher et al. (2007). For brevity, we use the term two-dimensional bin packing problem, or 2DBPP, henceforth. However, rather than focussing only on the packing problem alone, we also examine issues of production planning and scheduling.

In a classic 2DBPP, time is not considered an issue, and it is assumed that a rectangle can be allocated to any bin and the bins can be processed in any order. This is acceptable provided all the bins can be processed within a single production period. In this paper, we consider the issue of supplying the rectangles in a timely manner across multiple production periods, assuming that the capacity of the cutting process results in potential delays to the times at which the rectangles become available to the customer. More precisely, we assume that each stock sheet has a cutting time, and that each rectangle has an associated due date that specifies the time by which it should ideally be cut and available to the customer. As a result, the goal in this problem is to minimize the maximum lateness of the rectangles with respect to their due dates (DD) while using as few stock sheets as possible. This problem description captures the trade-off between using an ideal cutting with small waste which may result in some rectangles being delayed past their due dates, and using the due dates to group the rectangles for cutting which may result in more stock sheets being used than necessary.

There are few examples in the literature of implementations that tackle both the packing problem and the production planning/scheduling problem. These papers mainly focus on cutting-stock problems and tend to separate the problem of determining the cutting patterns and the scheduling of the parts. Gramani and França (2006) examine a similar problem, which can be viewed as one of the lot sizing, where pieces may be cut early and incur a holding cost but are not permitted to be late. They use a network shortest path model to generate a solution and deal with the packing element by selecting from available patterns. Nonas and Thorstenson (2000) also tackle the combined cutting-stock and

* Corresponding author. Tel.: +44 2380595671.
E-mail address: J.A.Bennell@soton.ac.uk (J.A. Bennell).

0925-5273/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ijpe.2013.04.040

lot-sizing problem, and additionally include setup times. They specifically look at steel parts for the production of trucks, which involves cutting irregular shapes. Hendry et al. (1996) combine the one-dimensional cutting-stock problem with production scheduling in the copper industry. They implement a two-stage procedure, where the first stage determines the number of logs to be produced and how they should be cut to minimize trim loss. The second stage determines a daily production schedule. Reinertsen and Vossen (2010) also consider the one-dimensional cutting stock problem with due dates where they seek to minimize tardiness. They identify a number of industries where this problem is important and assert that this combined problem is why manual scheduling is still wide spread. Their integer programming formulation considers production periods where the sequence of patterns within each period is arbitrary. For glass cutting, Puchinger et al. (2004) combine the optimization of the cutting pattern with the scheduling of loading the wagons with the cut pieces. Only three wagons are open at any one time and only pieces intended for the same customer may be loaded onto the same wagon. They implement two branch and bound approaches for optimizing the strip and bin subproblems, and compare these approaches with evolutionary algorithms. Arbib et al. (2012) consider a similar problem. They use a tabu search implementation to minimize the number of stock sheets required to cut a given set of pieces, while constraining the maximum number of open stacks allowed by the downstream buffers. They test their approach on the one dimensional stock cutting problem and claim that their approach is valid for the two dimensional case. Zheng et al. (2012) consider a cutting stock problem where the cutting pattern significantly impacts the production process efficiency. Large steel coils are cut into smaller rectangle pieces using horizontal and vertical cuts, transferring between the direction of cut has a significant overhead. Their heuristic seeks to maximize the weighted sum of material usage and processing efficiency. Li (1996) also considers a two-dimensional cutting stock problem for the manufacture of laboratory table tops. The mixed integer programming formulation assumes a finite set of known cutting patterns and schedules the patterns in earliest due date (EDD) order.

This paper proposes a genetic algorithm for solving the problem of cutting rectangles that have due dates, with the goal of minimizing the maximum lateness, while using as few bins as possible (henceforth, we refer to bins rather than stock sheets). We refer to this problem as the two-dimensional bin packing with due dates (2DBPP with DD). One contribution of this paper is addressing a problem that integrates the cutting and packing aspect with scheduling, hence taking account of how they impact on each other. For placing the rectangles in the bins, our genetic algorithm uses an adaptation of the *best fit* heuristic of Burke et al. (2004) for strip packing. This adaptation is our second contribution, where we show that the time complexity of Best Fit for Bin (BFB) packing is $O(n^2)$, where n is the number of rectangles, and compare its performance to recognized bin packing placement heuristics. BFB is sufficiently fast to be used repeatedly within a genetic algorithm, and is comparable in performance with existing heuristics that are computationally more expensive. The third contribution is an investigation into the use of a multicrossover approach within the genetic algorithm for bin packing. The multicrossover operator generates several candidate offspring before selecting two that are to be retained. Further investigation of this technique and comprehensive computational results that show the benefit of adopting the multicrossover approach which are provided by Lee (2006).

The remainder of this paper is organized as follows. In Section 2, we provide a formal description of the two-dimensional bin packing problem with rectangle due dates, and derive a lower bound on the maximum lateness. We review the relevant literature on two-dimensional bin packing in Section 3, including heuristic placement routines and local search methods. Sections 4 and 5 describe our placement heuristic, called the Best Fit Bin and the multicrossover genetic algorithm, respectively, for the 2DBPP with DD. In Section 6, we describe two neighbourhood search algorithms that are adapted from the Unified Tabu Search approach of Lodi et al. (1999a, 1999b, 2004) which are used to help benchmark the performance of our approach. We present our comprehensive computational results in Section 7, and some concluding remarks in Section 8.

2. Preliminaries

2.1. Problem definition

The problem we consider is most closely related to the non-orientated, two-dimensional, rectangular single bin size, bin packing problem. This problem can be defined as that of packing n rectangles into identical bins, each with height H and width W. Each rectangle j is defined by a width w_j and a height h_j, where $w_j \le W$ and $h_j \le H$, for $j=1,\ldots,n$. All rectangles must be packed so that their edges are parallel to the edges of the bins, although any rectangle j with $w_j \le H$ and $h_j \le W$ may be rotated by 90° before being packed into a bin. In a feasible packing, no rectangles should overlap with other rectangles, and all rectangles must be entirely contained within the bin. The goal is to minimize the number of bins used for the packing.

In our extended version of the problem, namely 2DBPP with DD, each rectangle j has a due date d_j, for $j=1,\ldots,n$, that defines the time by which the cutting of rectangle j should ideally be completed. Further, we assume that the cutting corresponding to any bin takes a constant time P. Thus, each rectangle j assigned to bin b has a completion time $C_j = bP$. The goal in this problem is to minimize the maximum lateness, $L_{\max}$. In order to calculate $L_{\max}$, we first calculate the due date of each bin as follows.

Let B be the number of bins used, and Q_b be the set of rectangles packed in bin b, for $b=1,\ldots,B$. Then the *bin due date* of bin b is defined by

$$\delta_b = \min_{j \in Q_b} d_j. \tag{1}$$

We re-index the bins in a non-decreasing order of their bin due dates, so that $\delta_1 \le \cdots \le \delta_B$. Thus, the maximum lateness of the rectangles packed is

$$L_{\max} = \max_{b=1,\ldots,B} \{bP - \delta_b\}. \tag{2}$$

However, we view the problem as a bicriteria optimization problem where the objective functions are to minimize the maximum lateness of the rectangles, and to minimize the number of bins used. The problem is NP-hard since the bin packing problem is NP-hard (Garey and Johnson, 1979), even with a common due date for all rectangles.

The motivation of this extension arises from the dilemma faced in the industrial manufacturing applications which involved the trade-off between the customers' satisfaction (meeting customers' due date on the order placed) and the manufacturer's efficiency (minimizing the wastage of material used).

When time is not considered as part of the packing problem, rectangles would typically be grouped into batches according to the date by which they must be cut followed by determining the cutting patterns while respecting the order of the batches. It is clear that in our approach by relaxing the due date constraint and including it within the objective function, removes the need for the initial grouping and allows the rectangles to be cut in any

order. Better packing arrangements may be found as a result of being able to select from a wider variety of rectangles when packing each bin. This clearly comes at the cost of potentially missing customer deadlines. However, depending on the perceived cost of disappointing a customer versus the cost of material, this option may be desirable. Further, this problem extension is of benefit when the capacity of the cutting process does not permit all due dates to be met and the manufacturer wishes to improve customer service by minimizing lateness.

The objectives of minimizing lateness and maximizing packing efficiency do not necessarily conflict. If mixing customer orders with different due dates provide better packing efficiency, which results in a reduction in the number of bins needed, then some orders are completed earlier (assuming a fixed bin processing time), albeit to the detriment of other orders that are completed later to facilitate the more efficient packing. Nevertheless, it is often possible to achieve a gain in packing efficiency so that some orders have a reduced lateness that more than compensates for other orders experiencing an increase in lateness.

2.2. Lower bound

In this section, we derive a simple lower bound on the maximum lateness for the 2DBPP with DD. The lower bound (for the number of bins used in non-oriented 2DBPP) proposed by Dell'Amico et al. (2002) is used in the derivation of this lower bound.

We first sort the n rectangles in earliest due date (EDD) order, so that $d_1 \leq \cdots \leq d_n$, and let $S_j = \{1, \ldots, j\}$ denote the subset of j rectangles with the smallest due dates. We compute a lower bound on the maximum lateness for the subset of rectangles S_j, which we denote by $\mathrm{LB}_{L_{\max}}(S_j)$, for $j = 1, \ldots, n$.

To compute $\mathrm{LB}_{L_{\max}}(S_j)$, we first evaluate the lower bound of Dell'Amico et al. (2002), which we denote by $\mathrm{LB}_{\mathrm{Bin}}(S_j)$, on the number of bins required to pack the rectangles of S_j. Thus, some rectangle must complete at time $\mathrm{LB}_{\mathrm{Bin}}(S_j)P$ or later, and therefore has a lateness of at least $\mathrm{LB}_{L_{\max}}(S_j) = \mathrm{LB}_{\mathrm{Bin}}(S_j)P - d_j$. Thus, our overall lower bound on the maximum lateness of the rectangles is given by

$$\mathrm{LB}_{L_{\max}} = \max_{j=1,\ldots,n} \mathrm{LB}_{L_{\max}}(S_j) = \max_{j=1,\ldots,n} \{\mathrm{LB}_{\mathrm{Bin}}(S_j)P - d_j\}. \quad (3)$$

3. Literature

Our review of the literature largely focusses on placement heuristics and local search methods for the two-dimensional bin packing problem. We exclude exact enumerative approaches and the worst-case analysis of approximation algorithms since such studies are not directly relevant to our work. Also, excellent and comprehensive reviews are provided by Dowsland and Dowsland (1992), Dyckhoff and Finke (1992), Lodi et al. (2002a, 2002b), and Hopper and Turton (2001a). Here we will discuss two key aspects of the bin packing literature: heuristic placement routines and local search approaches.

3.1. Heuristic placement routines

Classical placement heuristics work on levels, where the first level is at the bottom of the bin packed from left to right, and a new level is started at the top of the highest rectangle packed at the current level. Coffman et al. (1984) suggest three strategies; *next-fit*: pack the next rectangle on the current level if it fits, otherwise closing the level and packing on a new level; *first-fit*: pack the next rectangle on the first level where it fits, opening a new level when required; *best-fit*: pack the next rectangle on the level where it leaves minimum remaining horizontal space, opening a new level when required.

These level heuristics can be used for bin packing in two ways: either directly packing levels into a number of finite bins (one-phase), or packing levels onto a single bin of unbounded height (a strip) and then dividing the strip into finite bins (two-phase). The second strategy transforms the strip into a one-dimensional bin packing problem thereby allowing various classical heuristics to be applied.

Berkey and Wang (1987) compare the performance of the one- and two-phase approaches. For the level packing strategies, they implement next-fit and first-fit one-phase approaches, and first-fit and best-fit two-phase approaches. In all cases, the pieces are sorted in non-increasing height order. A comparison of the one- and two-phase versions of first-fit shows that the two-phase approach performs marginally better.

Lodi et al. (1999b) modify the level packing heuristics for both one- and two-phase approaches. For one-phase approach, instead of starting the next level on the left they alternate between left and right. Since the rectangles are ordered according to non-increasing height, this method of creating levels allows them to place the rectangles as low as possible. For the two-phase approach, referred to as Floor-Ceiling (FC) each level has a floor and ceiling defined by the bottom and top edge of the tallest rectangle packed on the level. Rectangles are then placed according to the priority list: (a) place on floor; (b) place on ceiling; and (c) open new level.

In the same study, Lodi et al. (1999b) describe a third approach called Touching Perimeter (TP). First $\mathrm{LB}_{\mathrm{Bin}}$ bins are opened where $\mathrm{LB}_{\mathrm{Bin}}$ is a lower bound. Each feasible packing position for the next rectangle is evaluated and the one with the largest percentage of the rectangle perimeter which touches either the bin or other rectangles is selected. If no feasible positions exist, then a new bin is opened. These three approaches are compared to the one-phase first-fit and two-phase best-fit methods of Berkey and Wang (1987). In most cases, all three routines outperform those of Berkey and Wang, with Touching Perimeter performing the best.

Bottom-Left (BL), introduced by Baker et al. (1980) and Jakobs (1996), is one of the best known placement heuristics across many different two-dimensional packing problem variants. The basic idea is to start from the top right corner of the bin, with each rectangle making successive moves of sliding as far as possible downwards and then as far as possible to the left until the rectangle is placed in a stable position. Liu and Teng (1999) modify this approach to generate the improved Bottom-Left approach. Instead of moving the rectangle the complete distance to the left, it moves the rectangle along the partial layout by giving downward movement priority. Bottom-Left Fill (BLF) proposed by Chazelle (1983) places rectangles by searching a list of location points that indicate potential positions where rectangles may be placed. Since the partial layout is based on the allocation of the lowest sufficiently large area, rather than on a series of bottom left moves, it is capable of filling existing gaps in the packing pattern. Compared to BL and improved BL, this method produces a denser packing. Finally, Best Fit (BF) by Burke et al. (2004) explicitly aims the fill the lowest gap in the layout prioritizing the tightest fit. Section 4 provides more details of the BF approach.

3.2. Local search methods

Local search methods provide a natural approach for instances where exact methods cannot solve the problem using reasonable computational resources. Genetic algorithms and tabu search appear to be the two most popular choices of local search techniques.

A common feature found in most genetic algorithms (GAs) developed for the 2DBPP is the use of the GA in combination with a heuristic placement routine. In the resulting two-stage approach, the GA manipulates the encoded solutions, which are then evaluated by a decoding algorithm that transforms a sequence of rectangles to be packed into the corresponding physical layout. The first researcher to use this approach was Smith (1985). He applies a GA to a two-dimensional rectangular packing problem with fixed orientation. The objective of his GA is to place as many rectangles as possible into a single rectangular bin. He uses permutations of rectangles to encode the solutions. Thus, the original problem becomes a sequencing problem, and heuristics are used to transform permutations into packing schemes. Computational results show that his GA can produce the same packing density 300 times faster than a dynamic program.

Jakobs' (1996) implementation for the strip packing problem uses a permutation chromosome that represents the packing sequence of the pieces, which is decoded using the BL placement heuristic. Crossover creates the first part of the child chromosome by replicating a portion of one parent chromosome, identified by a starting point and number of genes, and completes the chromosome by inserting the unrepresented genes in the order they appear in the second parent. Leung et al. (2001) use the same idea to tackle the cutting stock problem. They try a second placement heuristic called difference procedure and compare with a simulated annealing (SA) implementation. Their findings favor GAs. They make further developments of their GA in Leung et al. (2003) where the replacement strategy uses SA acceptance criteria to determine whether the child chromosome will enter the population, with the aim of avoiding premature convergence. Results show that this strategy only improved performance over a large number of generations.

Similar permutation representations are used by Hwang et al. (1994) and Hopper and Turton (1999) in their GA. However, they implement different decoding placement heuristics. Hopper and Turton (2001b) also compare a GA with simulated annealing (SA), naïve evolution (NE), hill climbing, and random search. Their computational results show that when BLF is used to decode the solution, GA, SA, and NE give similar quality solutions.

Lodi et al. (1999a, 1999b, 2004) develop a Unified Tabu Search (UTS) code for multi-dimensional bin packing problems. Their algorithm works directly with the constructed solution rather than with a representation. A *target bin* is identified, and at each iteration the algorithm attempts to move a rectangle j out of the target bin. There are two possible neighbourhood moves, with the first attempting to directly pack j into a different bin, and the second attempting to recombine the rectangles of two different bins so that one of them can accommodate j.

There are many interesting examples of GA implementations for other types of two-dimensional rectangular packing problems. Kroger (1995) included guillotine constraints directly in the encoding of the chromosome. The author describes a slicing tree representation of a guillotine layout where the leaf nodes are the pieces and the other nodes define the guillotine cuts, either horizontal or vertical. The chromosome replicates the slicing tree depth first, listing the hierarchy of cuts and pieces, backtracking once a leaf node is reached. Crossover and mutation operate on subtrees in the chromosome. Beasley (2004) and Gonçalves and Resende (2011) both look at output maximization problems. In both cases their chromosomes have two parts. In the case of Beasley (2004), the first part is a binary encoding and identifies whether a piece is placed or not, the second part is the co-ordinate position of the piece in the layout. Infeasibility is permitted and penalized in the fitness function. Gonçalves and Resende (2011) adopt the packing sequence strategy represented in the first part of the chromosome, which is decoded by a sequence of placement rules, held in the second part of the chromosome. Rather than directly working on a permutation, they use a random key encoding that ensures that any crossover is feasible.

4. Best Fit Bin

As in previous genetic algorithm implementations, we will use a placement heuristic to decode the gene. We adopt the BF heuristic placement routine developed by Burke et al. (2004). Since this heuristic was designed for the strip packing problem and we are solving the bin packing problem, we have made some adaptations. We call the placement heuristic Best Fit Bin (BFB).

BF aims to fill the available gaps in the partial layout by dynamically selecting the best rectangle for placement during the packing stage. Unlike the Bottom-Left (BL) and BLF approaches that place the rectangles based on the sequence of rectangles supplied, BF makes informed decisions about which rectangle to pack next and where it should be placed. Extensive computational results of Burke et al. (2004) indicate that high-quality packings are generated by BF.

Based on the ideas in BF, we propose the BFB placement routine for the two-dimensional bin packing problem. This placement routine consists of two stages: *preprocessing* stage and *packing* stage. As in BF, the preprocessing stage arranges each rectangle in a horizontal orientation where its longest edge is parallel to the bottom of the bin, and sorts the rectangles in non-increasing order of their width (breaking ties by non-increasing height). The packing stage follows the same principle as BF for identifying the lowest gap where a piece might be placed. BF also includes a postprocessing phase that is not appropriate for the BPP.

Unlike the BL and BLF approaches, BFB packs the bins one at a time, closing the current bin and initializing a new bin whenever none of the remaining rectangles will fit within any of the available gaps in the current bin. The packing stage employs a best-fit type strategy by examining the lowest available gap in the current bin and then placing the rectangle that best fits the gap available. This placement routine not only keeps track of the free positions in the layout, but also of the dimensions of the available gap at the respective positions.

We use a coordinate system to describe the profile of available gaps in the current bin (Burke et al., 2009). The bottom left of the bin has a coordinate (0,0) and the top right of the bin has a coordinate (W,H). Let $(x_1,y_1),\ldots,(x_r,y_r)$ be coordinates which define the profile of packed and wasted space and potentially usable space, where $0=x_1<\cdots<x_r<W$ and $0\le y_q\le H$ for $q=1,\ldots,r$. Specifically, the profile is defined by lines joining the points (x_1,y_1) and (x_2,y_1), (x_2,y_2) and $(x_3,y_2),\ldots,(x_{r-1},y_{r-1})$ and (x_r,y_{r-1}), and (x_r,y_r) and (W,y_r). An example of the coordinate system is illustrated in Fig. 1. All space in the bin below these lines is either packed with

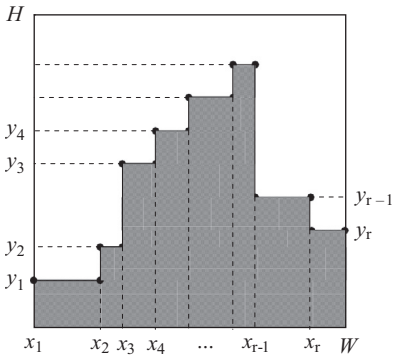

Fig. 1. Example of a profile defined by the coordinate system.

rectangles or is wasted space because the rectangles are too large to fit, while the space above the lines is available to be used for further packing.

Using the profile, the value $y_q = \min_{p=1,\ldots,r} y_p$ defines the lowest gap, and we can compute its width as $g_w = x_{q+1} - x_q$, where $x_{r+1} = W$, and its height as $g_h = H - y_q$. For a rectangle j with $w_j \le g_w$ and $h_j \le g_h$, its fill is defined by $f_j = w_j$. Similarly, by considering a rotation of rectangle j, if $h_j \le g_w < w_j$ and $w_j \le g_h$, its fill is $f_j = h_j$. The Best Fit Bin strategy selects a candidate rectangle j with the largest f_j, breaking ties by selecting a rectangle with the largest area. If there are no candidates for the gap, a dummy rectangle corresponding to wasted space is inserted into the gap.

A formal description of Algorithm BFB is given below.

Algorithm BFB.

Step 1. Orient the rectangles so that $w_i \ge h_i$, and index so that $w_1 \ge \cdots \ge w_n$, and $h_i \ge h_{i+1}$ if $w_i = w_{i+1}$ for $i = 1, \ldots, n-1$.

Step 2. Initialize a bin and assign the first rectangle, j, in the list to that bin. Define the profile: $(x_1, y_1) = (0, h_j)$. If $w_j < W$, then $(x_2, y_2) = (w_j, 0)$.
Remove j from the list of rectangles.

Step 3. Choose the smallest index q such that $y_q = \min_{p=1,\ldots,r} y_p$.
- If $y_q = H$, then close the current bin and go to Step 2.
- Otherwise, compute the gap width g_w and the gap height g_h.

Step 4. Scan the list of rectangle to find the best fit.
- For the current rectangle i, set $f_i = 0$ if $w_i > \max\{g_w, g_h\}$ or if $h_i > \min\{g_w, g_h\}$ and then proceed to the next rectangle.
- If $w_i > g_w$, then set $f_i = h_i$; otherwise, set $f_i = w_i$
- If all rectangles are scanned and $\max_i f_i = 0$, go to Step 6; otherwise, choose j such that $f_j = \max_i f_i$ (breaking ties by area), and go to Step 5.

Step 5. If $w_j > g_w$, then interchange w_j and h_j to change the orientation of rectangle j.
Place rectangle j with its bottom left corner at the coordinate (x_q, y_q).
Remove j from the list of rectangles, Go to Step 7.

Step 6. Create a dummy rectangle 0 with width $w_0 = g_w$, and height given by the minimum height of the adjacent rectangles, or the remaining height of the bin when $g_w = W$.
Insert waste by placing rectangle 0 with its bottom left corner at the coordinate (x_q, y_q), and go to Step 7.

Step 7. Update the profile.
If not all rectangles are packed, go to Step 3; otherwise terminate.

Step 1 is a preprocessing step that facilitates efficient subsequent searches for a rectangle with the largest fill. Step 2 initiates a new bin, and places the first rectangle on the list in the bottom left of the bin in accordance with the Best Fit Bin strategy. The lowest gap is identified in Step 3, and its width and height are computed. Step 4 executes the search for a candidate rectangle j with the largest f_j. Since the rectangles are searched in list order, a key observation is that when a rectangle fits within the gap without reorientation, then the search stops, since any other rectangle cannot produce a larger fill (or the same fill and have a larger area). When there is a candidate rectangle that can only be placed by reorienting it, then it is worthwhile for the search to continue. Any rectangle that exactly fills the gap is immediately placed. On finding a rectangle that fits within the gap without reorientation, the search can stop and the rectangle with the higher fill is immediately placed. Step 5 is executed when a suitable candidate rectangle for the gap is found. On the other hand, when there is no candidate, Step 6 creates a dummy rectangle that corresponds to wasted space. Finally, Step 7 updates the profile that results from placing the candidate rectangle j or the dummy rectangle.

We now analyze the time complexity of Algorithm BFB.

Theorem 1. *Algorithm BFB requires $O(n^2)$ time.*

Proof. Step 1 of Algorithm BFB requires $O(n\log n)$ time to orient and order the rectangles. To analyze the remaining steps, suppose that the final packing uses B bins to pack all of the rectangles, where $B \le n$, and that m_b rectangles are packed into bin b, for $b = 1,\ldots,B$, where $\sum_{b=1}^{B} m_b = n$. Step 2 is executed B times and therefore requires $O(n)$ time.

Next we analyze how many times Step 3 is executed. Note that packing a gap with an actual rectangle adds at most one coordinate to the profile, while packing a gap with a dummy rectangle removes at least one coordinate from the profile. Therefore, for bin b in which m_b actual rectangles are packed, there are at most $m_b + 1$ dummy rectangles. Since the first rectangle of a bin is placed immediately in Step 2, it follows that Step 3, and consequently Steps 4, 5, 6 and 7 are executed at most $2m_b$ times for bin b, and at most $2n$ times in total. Further, Steps 3 and 4 require at most $O(n)$ time per execution, while Steps 5, 6 and 7 require constant time per execution. Thus, Steps 3 through 7 require $O(n^2)$ time overall.

Combining the above, we obtain a time complexity of $O(n^2)$ for Algorithm BFB. □

5. Multicrossover genetic algorithms

In this section, we describe our proposed genetic algorithm (GA) implementation. For a comprehensive discussion of GAs see Goldberg (1989). A key feature of our approach is that we apply the crossover multiple times to each set of parents in order to optimize the quality of the child chromosomes. We call this approach as the multicrossover genetic algorithm (MXGA). Examples of this approach can be found in the computer science literature. Esquivel et al. (1997) present an empirical investigation of generating up to six offspring from two parent solutions. All offspring enter the next generation. Herrera et al. (2002) investigate a much larger number of offspring per mating pair and select the best two for the next generation. Both papers use benchmark test functions and both reports improved performance with the multiple crossover GA along with diminishing returns for increasing the number of offspring. The general framework of the MXGA is explained below with specific reference to the 2DBPP. A more detailed discussion of this approach is given by Lee (2006)

5.1. Representation

In our proposed MXGA, each chromosome is of length n and corresponds to the rectangles. The genes are integers from the set $\{1, \ldots, \mathrm{LB}_{\mathrm{Bin}}\}$ with each gene indicating the bin into which the corresponding rectangle is to be packed, where $\mathrm{LB}_{\mathrm{Bin}}$ is the lower bound of Dell'Amico et al. (2002). A solution to the packing problem is therefore represented by a sequence of positive integers indicating the bin numbers for the rectangles, where the exact location in the layout is then determined by applying our decoding scheme that utilizes BFB as described in Section 4.

Fig. 2 shows an example of the gene representation for an individual, defining an attempt to pack 8 rectangles into 3 bins ($n = 8$ and $\mathrm{LB}_{\mathrm{Bin}} = 3$). An attempt is made to pack rectangles {3,5,7} into bin 1, rectangles {1,4} into bin 2, and rectangles {2,6,8} into bin 3. Any rectangle that cannot be feasibly packed into their assigned bins is dealt with a repack strategy to be explained later.

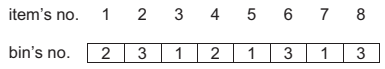

Fig. 2. An example of an individual (chromosome)

5.2. Decoding

The BFB Algorithm for placing the rectangles is used to decode the genotype of an individual into a phenotype (packing layout). BFB produces a valid packing, but there may be rectangles that remain unpacked.

During the process of packing the rectangles into a bin, any rectangle that cannot be feasibly packed will be regarded as an unassigned rectangle and kept in a list. After BFB has been applied to all of the bins $1,\ldots,\mathrm{LB}_{\mathrm{Bin}}$, a *repack* strategy is employed to pack any rectangles in the unassigned rectangle list into the bins already used or into new bins, so all rectangles are feasibly packed. After repacking, the genotype of the individual is updated so that it corresponds to the new packing layout.

The *utilization* of a bin is defined as the total area of rectangles packed into the bin divided by the total area WH of the bin. We now present our repacking algorithm.

Algorithm Repack.

*Step*1. Form a list of bins ordered in non-decreasing order of utilization.
Form a list of the unassigned rectangles ordered in non-increasing area.
*Step*2. Consider the next bin in the list.
*Step*3. Attempt to repack the selected bin using Algorithm BFB considering all of its rectangles plus the next rectangle in the unassigned list. Any rectangle not packed becomes unassigned at this stage. Compute the new bin utilization.
(a) If the bin utilization has increased, accept the new packing layout and update the unassigned rectangle list. If the unassigned rectangle list is empty, then stop; otherwise, go to Step 1.
(b) Otherwise, the new packing layout is rejected.
- If not all rectangles are considered, then proceed to the next rectangle.
- If all rectangles in the unassigned list are considered and the selected bin is not last, then go to Step 2.
- If all rectangles in the unassigned list are considered and the selected bin is last, then go to Step 4.

*Step*4. Pack any unassigned rectangle in the list into one or more new bins using Algorithm BFB until all rectangles are packed.

5.3. Initial population

The initial population is generated by randomly choosing each gene from the set $\{1,\ldots,\mathrm{LB}_{\mathrm{Bin}}\}$, where $\mathrm{LB}_{\mathrm{Bin}}$ is the lower bound of Dell'Amico et al. (2002). The population size remains constant throughout the algorithm.

5.4. Selection mechanism

We use a probabilistic binary tournament selection scheme as the selection mechanism of each parent in the MXGA. As the name suggests, two individuals are chosen at random from the population, and then a random number r is generated from the uniform distribution defined on the interval $[0,1]$. If $r<s$, where s is a parameter, the fitter of the two individuals is selected to be the parent; otherwise, the less fit individual is selected. The two individuals are then returned to the original population and can be selected again.

5.5. Multicrossover operator

The multicrossover operator in MXGA has the simple 1-point or 2-point crossover at its core. Instead of performing a single 1-point or 2-point crossover with each pair of parents, the crossover process is repeated t times, to produce $2t$ temporary offspring. Each offspring is then decoded using BFB. In order to reduce computational time in this stage, the unallocated list of rectangles are packed into new bins rather than using the more expensive Repack procedure. Two temporary offspring are selected; the best and one other chosen through a probabilistic binary tournament. Note that the crossover operator is only applied to the selected parents with a given *crossover* probability p_c. When crossover is not applied, we apply the swap operation that is described below.

5.6. Swap operator

The use of a crossover probability p_c dictates that multicrossover may not be applied to the selected parents. Since we use an elitist replacement strategy where all parents and children compete for a place in the next generation, there is no value in making an exact duplicate of the parents. As a result, a new operator called *swap* is used in the MXGA to produce two offspring that are different from their parents. By doing this, we introduce more diversity to the search space. The basic step of this operator is to randomly select a swap point in a parent, and then swap the substrings separated by the swap point to form an offspring.

5.7. Mutation operator

We apply the mutation operator in the MXGA in two stages. First, a subset of individuals is selected from the new offspring population with a given *individual mutation* probability p_M. Then each gene in the selected offspring is considered in turn, and the bin number is randomly changed to an element of the set $\{1,\ldots,\mathrm{LB}_{\mathrm{Bin}}\}$ according to the *gene mutation* probability p_m.

5.8. Fitness evaluation

As described earlier, our problem has two objectives: minimizing number of bins used to pack the complete set of rectangles, and minimizing the maximum lateness. Next, we discuss each objective separately.

The number of bins used to pack the complete set of rectangles is straightforward. However, by itself minimizing this value is not sufficient to guide the search process since a large number of solutions use the same total number of bins but may be of very different quality with respect to the evolution of the search. This suggests the use of a secondary fitness function to break ties. Based on the observation that the quality of a packing pattern can also be evaluated by the utilization of the bin, Falkenauer and Delchambre (1992) suggest a fitness function: $F=\sum_{b=1}^{B}U_b^{\nu}/B$, where B is the total number of bins used, U_b is the utilization of bin b for $b=1,\ldots,B$, and ν is a parameter with value $\nu>1$. They experimented with several values for ν and suggest that $\nu=2$ is an appropriate value. The exponent means that a combination of high and low utilization bins is preferred to evenly distributed utilization. Since we are seeking to minimize the number of bins, and so need to empty existing bins, this fitness function is useful. In order to give greater preference to a single weak bin, we modify the (secondary) fitness function by first re-indexing the bins so that $U_1\geq\cdots\geq U_B$, and then computing the fitness in the same way as Falkenauer and Delchambre (1992), while ignoring the last bin.

For the maximum lateness objective function, recall that the maximum lateness is computed using (1) and (2). We use the value of L_{max} as our fitness function for the maximum lateness objective.

In order to deal with the two objectives defined for the 2DBPP with DD, we use the lexicographical ordering approach and use both possible orderings. Lexicographic preferences give an ordered value of each objective. The search will seek to optimize the objective function appearing first, and then the objective function appearing second is optimized subject to the additional constraint that the solution value of the first objective does not deteriorate. Lexicographic ordering of objectives where preferences can be articulated prior to the optimization is a well known approach (Evans, 1984). Since we do not have a known preference we alternate between the two possible orderings: minimize the maximum lateness of the rectangles and then the number of bins (ideal for customers' satisfaction), minimize the number of bins and then the maximum lateness (ideal for manufacturer's efficiency). Every G generations we exchange the order of the objectives and continue the search keeping track of all the non-dominated solutions. In most cases this will include a solution that has the lowest L_{max} for each value of B between the smallest B found and the value of B for the solution with lowest L_{max}. With this approach we aim to find good solutions that represent the trade-off between the customers' satisfaction and manufacturer's efficiency.

5.9. Replacement and filtration strategies

Our proposed MXGA uses the elitism replacement scheme whereby the offspring have to compete with their parents to gain admission to the new population. In the elitism replacement stage, combine both parent and offspring population into a single population of size $2n_{pop}$, which are sorted in a non-increasing order of their associated fitness. Select the first half for the next generation. Since we alternate between different fitness functions every G iterations, we also store the best solutions found for each objective so far in a separate archive.

After n_{pop} individuals have been selected, a process called *filtration* is used to identify the identical individuals from the new population. Any duplicate individuals are removed and replaced by uniformly randomly generated new individuals to avoid premature convergence and to add diversity to the new population. Because of the computational cost of the filtration procedure, we invoke this procedure every R generations.

6. Neighbourhood search algorithms

Since we are tackling a new variant of the 2DBPP, it is useful to develop competing algorithms to act as a benchmark to evaluate MXGA. This section describes two competitors, one based on the Unified Tabu Search (UTS) of Lodi et al. (1999a, 1999b, 2004), and a Randomized Descent Method (RDM) that adopts many of the features of the UTS. Modifying the UTS approach to include due dates is a natural choice for a tabu search algorithm since UTS generates some of the best solutions for the 2DBPP. For more details about the framework of UTS, we refer to Lodi et al. (1999a, 1999b, 2002a, 2004). For the 2DBPP with DD, we alternate the objective functions (as discussed in Section 5.8) every I iterations, and employ *first improve* strategy for RDM. Two neighbourhoods are used depending on which objective function is currently under consideration.

6.1. Unified Tabu Search

The Unified Tabu Search (UTS) algorithm of Lodi et al. (1999a, 1999b, 2004), which we refer to as UTS_{TP}, uses TP as a heuristic placement routine. In order to make a clear comparison between UTS and MXGA, our version of UTS uses the BFB placement method, and is denoted by UTS_{BFB}. We describe below the two alternative versions of UTS_{BFB} depending on whether the main objective is to minimize the number of bins or the maximum lateness. In both versions we attempt to follow the methodology of Lodi et al. (1999b) as closely as possible.

We first describe the version of UTS_{BFB} for minimizing the number of bins. For a given solution, the search identifies a target with the aim of emptying this bin. The target bin is selected according to the number and value of the rectangles it contains, which is quantified by a filling function. The neighbourhood is defined by the removal of a rectangle j contained in the target bin, and the repacking of j together with contents of k other bins. More precisely, k bins are emptied and the removed rectangles along with j are repacked. The neighbourhood considers each rectangle j in the target bin and each k-tuple of bins. There are k tabu lists, one for each neighbourhood size.

The value of k varies dynamically as the algorithm progresses. When rectangle j is successfully moved and the total number of bins is reduced, the value of k is decreased by one. If the number of bins remains the same, then the move is only accepted, and k reduced, if the move is non-tabu; otherwise, the move is assigned a penalty of infinity. Finally, if the move increases the number of bins (i.e. to $k+1$), then a subproblem is solved to determine whether the move is rejected. The subproblem involves identifying a new target bin out of the $k+1$ bins, emptying this bin, and repacking the rectangles along with rectangle j. If these rectangles fit into a single bin, then the penalty is the minimum filling function value of the $k+1$ bins; otherwise, the penalty is infinity. Once the entire neighbourhood for the target bin has been searched, the move with the minimum finite penalty is accepted with k remaining unchanged. If there is no move with a penalty less than infinity, then k is increased by one. An upper limit is set for k, and if this limit is reached, then diversification moves are performed. Each neighbourhood size has a tabu list where the minimum penalty for the last τ neighbourhoods are stored, thus preventing cycling. We have implemented this algorithm using BFB for placing the rectangles in the bins, an upper limit of 3 for k, and $\tau = 3$.

We now turn our attention to the version of UTS_{BFB} that is designed for the objective of minimizing the maximum lateness. Only relatively minor modifications are needed to deal with the alternative objective function. The maximum lateness of the current solution is found, as usual, by using Eq. (1), renumbering, and applying Eq. (2). Then, we define the target bin to be a bin containing a rectangle with the maximum lateness.

The moves attempt to remove the rectangle(s) with the smallest due date from the target bin in the same way as the original UTS. However, in this case our objective is to minimize the maximum lateness. A potential move is one in which the rectangles from the k selected bins, plus the rectangle from the target bin are repacked. The resulting solution is accepted if: (i) the maximum lateness over all bins is decreased, or (ii) the number of bins used does not exceed the current solution value while maintaining the maximum lateness value. Note that we allow the number of bins used to increase if it results in a decrease in the maximum lateness.

As in the original procedure, the value of k is updated during the execution of the algorithm. When either (i) or (ii) is applied, the move is immediately performed, and k is reduced by one. If the neighbourhood has been completely searched without finding a

move that is accepted, then k is increased by one unless an upper limit is reached. In the latter case when k reaches the upper limit, a diversification procedure is performed. When neither (i) nor (ii) apply, a penalty is associated with the move. The penalty is infinity if the move is tabu, or if the maximum lateness value obtained from the new packing is higher than the current solution. If the search of the neighbourhood is completed without detecting cases (i) and (ii), then a move having the minimum finite penalty (if any) is performed. As for the original UTS, there is a tabu list and a tabu tenure for each value of k. Each list stores the penalty values corresponding to the previous moves performed.

6.2. Randomized Descent Method

The Randomized Descent Method (RDM) that we implement uses the same neighbourhoods and diversification procedure as those in UTS. The main difference lies in the removal of the tabu list and an alternative acceptance rule. When the objective is to minimize the number of bins, if the target bin is emptied then this is an improving move and is accepted. If the move fails to empty the bin, even if some items are removed, this is a neutral move. If the number of bins is increased then this is a deteriorating move and rejected. In the case of minimizing the maximum lateness, only the rectangles with the smallest due dates must be removed and repacked elsewhere. A decrease in maximum lateness is an improving move and accepted, even if the number of bins is increased, the same maximum lateness while maintaining the number of bins is a neutral move, otherwise it is a deteriorating move and rejected. As the search progresses, emptying an entire bin, or moving the maximum lateness item to a bin that is then re-ordered by the EDD rule to improve the lateness becomes quite challenging. We found that there are many neutral moves forming plateaux areas in the solution space. Hence the acceptance rule in RDM allows up to 1000 neutral moves for consecutive iterations before terminating the algorithm. When there are multiple identical neutral moves found during the neighbourhood search procedure in a single iteration, we randomly select one of these moves. Consequently, the procedure can negotiate these plateaux. Note that deteriorating moves are automatically rejected, rather than solving the subproblem involving assigning penalties as in UTS.

7. Computational experience

In this section, we explain the experimental design used in the evaluation of our algorithms, and present and discuss the results of our computational experiments. The first set of experiments compares BFB with some well-known heuristic placement routines, namely BLF, Touching Perimeter (TP) and Floor-Ceiling (FC). We then compare our proposed MXGA with a Single Crossover Genetic Algorithm (SGA) that is identical to MXGA except for the crossover, UTS and RDM for the standard 2DBPP, and also for the 2DBPP with rectangle due dates.

7.1. Experimental design

The algorithms are coded in ANSI-C using Microsoft Visual C++ 6.0 as the compiler, and run on a Pentium 4, 2.0 GHz computer with 2.0 GB RAM. We consider ten different classes of problem instances that have formed the basis for comparing algorithms in previous studies reported in the literature. These classes are listed in Table 1. The first six classes (I–VI) are introduced by Berkey and Wang (1987), while the other four classes (VII–X) are introduced by Martello and Vigo (1998) and are based on the following types of rectangles that are defined in terms of the width W and height H of the bins.

Table 1
Data types for the problem instances (Lodi et al., 1999b).

Data class	Bin ($W \times H$)	Item (w_j and h_j)
I	10×10	uniformly random in [1,10]
II	30×30	uniformly random in [1,10]
III	40×40	uniformly random in [1,35]
IV	100×100	uniformly random in [1,35]
V	100×100	uniformly random in[1,100]
VI	300×300	uniformly random in [1,100]
VII	100×100	Type 1 with probability 70%, Type 2, 3, 4 with probability 10% each
VIII	100×100	Type 2 with probability 70%, Type 1, 3, 4 with probability 10% each
IX	100×100	Type 3 with probability 70%, Type 1, 2, 4 with probability 10% each
X	100×100	Type 4 with probability 70%, Type 1, 2, 3 with probability 10% each

Type 1: w_j uniformly random in $[\frac{2}{3}W, W]$; h_j uniformly random in $[1, \frac{1}{2}H]$
Type 2: w_j uniformly random in $[1, \frac{1}{2}W]$; h_j uniformly random in $[\frac{2}{3}H, H]$
Type 3: w_j uniformly random in $[\frac{1}{2}W, W]$; h_j uniformly random in $[\frac{1}{2}H, H]$
Type 4: w_j uniformly random in $[1, \frac{1}{2}W]$; h_j uniformly random in $[1, \frac{1}{2}H]$.

For each class, the five values $n=20$, 40, 60, 80, 100 are considered. Also, for each combination of class and value of n, 10 problem instances are generated. The problem instances are provided by Lodi et al. (1999b) and are publicly available (www.or.deis.unibo.it/research.html)

In order to evaluate the algorithms' performance, we use the lower bound LB_{Bin} proposed by Dell'Amico et al. (2002) for the number of bins, and the lower bound $LB_{L_{max}}$ derived in Section 2.2 for the maximum lateness, where the bin processing time is $P=100$. A lower bound on the completion time of the last bin is given by PLB_{Bin}. For each problem instance (assuming $LB_{Bin} > 1$), we generate three sets of integer due dates from the uniform distribution of $[101, \beta PLB_{Bin}]$, where $\beta \in \{0.6, 0.8, 1.0\}$. We label each set of due date class as Class A for $\beta = 0.6$, Class B for $\beta = 0.8$ and Class C for $\beta = 1.0$.

We compare the performance of the various heuristic placement routines and local search algorithms on the basis of the Relative Percentage Deviation for Bins, Mean Squared Utilization of the bins, the Relative Percentage Deviation for maximum Lateness, defined by

$$\text{RPD}: \text{B} = \frac{100(\text{UB}_{\text{Bin}} - \text{LB}_{\text{Bin}})}{\text{LB}_{\text{Bin}}}, \tag{4}$$

$$\text{MSU} = \frac{\sum_{j=1}^{\text{UB}_{\text{Bin}}} U_j^2}{\text{UB}_{\text{Bin}}}, \tag{5}$$

$$\text{RPD}: \text{L} = \frac{100(\text{UB}_{L_{\max}} - \text{LB}_{L_{\max}})}{\text{LB}_{L_{\max}}}, \tag{6}$$

where UB_{Bin} and $UB_{L_{max}}$ represent the heuristic solutions found for the number of bins used and the maximum lateness, respectively, and LB_{Bin} and $LB_{L_{max}}$ are the lower bounds. The use of squared utilization as a performance measure in (5) is consistent with its use within our fitness function. The tables of results provided below report the average of the 10 instances generated for each due date class and each value of n.

Table 2
Values of average RPD:B for placement heuristics.

Data	n	Placement heuristic			
Class		BLF	BFB	FC	TP
I	20	9	3	6	5
	40	12	4	8	6
	60	13	5	9	5
	80	15	6	9	6
	100	12	4	7	3
	All	12	**4**	8	5
II	20	0	0	0	0
	40	10	10	10	10
	60	10	5	5	0
	80	7	7	3	7
	100	6	3	3	0
	All	7	5	4	**3**
III	20	20	6	18	6
	40	22	13	16	11
	60	26	10	19	11
	80	27	10	15	10
	100	23	8	13	8
	All	24	9	16	**9**
IV	20	0	0	0	0
	40	0	0	0	0
	60	10	15	10	10
	80	10	10	10	7
	100	13	7	7	3
	All	7	6	5	**4**
V	20	15	9	8	6
	40	18	10	10	11
	60	16	9	11	8
	80	17	9	11	8
	100	16	8	10	8
	All	16	9	10	**8**
VI	20	0	0	0	0
	40	40	40	40	40
	60	10	5	5	5
	80	0	0	0	0
	100	13	7	7	7
	All	13	**10**	10	10
VII	20	22	19	19	13
	40	20	12	17	10
	60	20	10	18	12
	80	20	10	17	11
	100	19	9	17	11
	All	20	12	18	**11**
VIII	20	23	15	16	16
	40	22	16	19	16
	60	19	9	18	11
	80	19	10	16	11
	100	19	9	17	12
	All	20	**12**	17	13
IX	20	1	1	0	1
	40	2	2	1	2
	60	1	1	1	1
	80	1	1	1	1
	100	1	1	1	1
	All	1	1	**1**	1
X	20	15	20	15	20
	40	13	7	9	8
	60	14	8	9	9
	80	14	6	6	6
	100	11	7	7	6
	All	13	10	**9**	10
Overall		13	8	10	**8**

Parameter settings for our proposed MXGA are determined through the results of initial computational experiments, as described by Lee (2006). We compare $t=3,5,7,9,10$ for the size of the candidate list of temporary offspring in MXGA (see Section 5.5) with $t=5$ providing the best quality solutions within a reasonable computation time. The crossover probability and mutation probability (see Sections 5.5 and 5.7) are varied, resulting in the settings $p_c=0.75$, $p_M=0.25$ and $p_m=1/n$. These values keep the mutation on a moderate scale while introducing small changes in the selected offspring. Potential parents are selected (see Section 5.4) by a binary tournament. Based on initial experiments, we set $s=0.75$, which gives a 75% chance of selection as the parent for the fitter individual as compared to the less fit individual which only has a 25% chance. The size of the initial population is set as $n_{pop}=100$ (see Section 5.3), and the filtration procedure (see Section 5.9) is invoked every 50 generations. Results showing the impact on performance of the addition of the swap operator and the mutation operator (see Sections 5.6 and 5.7) is also provided by Lee (2006).

7.2. Comparison of heuristic placement routines

In this subsection, we compare our proposed Best Fit Bin (BFB) with Bottom Left Fill (BLF), Floor-Ceiling (FC) and Touching Perimeter (TP). Previous studies with BLF suggest that ordering the rectangles by non-increasing width, height or area improves the quality of the resulting solutions. Our experiments show that ordering by decreasing area, breaking ties by decreasing width, provides the best solution quality.

Table 2 reports the average RPD:B for each placement routine, computed over the 10 problem instances. The values for FC and TP are quoted from the study of Lodi et al. (1999b). The final row for each data class gives the overall average values of RPD:B, and similarly the final row of the table gives the overall average values of RPD:B over all classes. The embolding highlights the best average RPD:B found in each class, and overall. Note that we do not list the computation times for these runs since they are very small (less than 0.1 s for any instance).

Examining the results in Table 2, we observe that BLF placement routine consistently produces the worst solutions among the four routines tested. Neither of the placement routines BFB, FC or TP can be classified as the clear winner in these tests as they produce mixed degrees of success in terms of the solution quality in each data class. For the average value over all classes, BFB and TP perform equally well, and BFB outperforms FC by 2%.

Given the similar quality solutions that are generated by BFB, FC and TP, and the time complexity of BFB being only $O(n^2)$ compared to $O(n^3)$ for FC and TP, it is advantageous to employ BFB as our heuristic placement routine within a genetic algorithm or local search procedure for the 2DBPP.

Table 3
Values of average RPD:B and average MSU for local search algorithms for 2DBPP.

Data class	*n*	SGA				MXGA				UTS_{BFB}		UTS_{TP}	RDM	
		1P		2P		1P		2P						
		RPD:B	MSU	RPD:B	MSU	RPD:B	MSU	RPD:B	MSU	RPD:B	MSU	RPD:B	RPD:B	MSU
I	20	2.7	81.3	2.7	81.2	2.7	81.7	2.7	81.9	2.7	81.3	5.0	4.7	79.9
	40	3.8	86.0	4.9	84.5	3.8	86.2	2.9	87.2	3.6	85.6	4.0	4.6	83.5
	60	5.2	86.7	4.0	88.0	4.0	88.6	4.0	88.5	4.0	88.1	4.0	4.0	88.2
	80	5.1	87.0	5.9	87.2	4.3	88.2	5.1	87.8	5.9	87.3	6.0	5.9	86.8
	100	2.8	92.4	1.9	92.7	2.1	93.6	2.5	92.9	3.1	92.5	3.0	3.1	92.2
	All	3.9	86.7	3.9	86.7	**3.4**	**87.7**	**3.4**	87.6	3.9	87.0	4.4	4.5	86.1
II	20	0.0	42.4	0.0	42.4	0.0	42.4	0.0	42.4	0.0	42.4	0.0	0.0	42.4
	40	10.0	54.8	10.0	55.0	10.0	56.1	10.0	56.1	10.0	56.1	10.0	10.0	56.0
	60	20.0	67.2	16.7	70.4	0.0	76.5	0.0	76.8	0.0	76.8	0.0	0.0	76.8
	80	6.7	78.0	6.7	78.0	0.0	83.5	3.3	81.1	0.0	83.5	3.0	3.3	81.3
	100	3.3	78.2	3.3	78.1	0.0	81.5	0.0	81.4	0.0	81.5	0.0	0.0	81.5
	All	8.0	64.1	7.3	64.8	**2.0**	68.0	2.7	67.5	**2.0**	**68.0**	2.6	2.7	67.6
III	20	5.7	66.4	5.7	66.5	3.7	69.0	3.7	69.4	3.7	68.9	6.0	5.7	66.2
	40	8.9	73.4	10.6	71.4	8.9	74.4	8.9	74.8	8.9	74.6	9.0	9.7	73.0
	60	10.3	77.8	9.5	78.5	7.9	82.0	8.7	81.0	9.5	80.5	8.0	9.5	80.5
	80	9.3	79.4	9.2	79.4	6.3	83.4	6.3	83.4	9.2	79.6	7.0	9.4	79.2
	100	8.2	81.2	8.7	80.6	5.7	85.3	6.2	84.3	6.2	83.5	7.0	7.2	82.8
	All	8.5	75.6	8.7	75.3	**6.5**	**78.8**	6.8	78.6	7.5	77.4	7.4	8.3	76.3
IV	20	0.0	38.4	0.0	38.4	0.0	38.4	0.0	38.4	0.0	38.4	0.0	0.0	38.4
	40	0.0	55.1	0.0	55.0	0.0	56.7	0.0	56.6	0.0	56.7	0.0	0.0	56.7
	60	10.0	70.1	10.0	69.6	10.0	71.5	10.0	71.3	10.0	71.1	10.0	10.0	71.1
	80	10.0	73.0	10.0	72.9	6.7	76.5	10.0	74.2	3.3	79.3	7.0	3.3	79.3
	100	6.7	75.8	6.7	75.4	3.3	79.3	3.3	79.0	3.3	79.2	3.0	3.3	79.2
	All	5.3	62.4	5.3	62.3	4.0	64.5	4.7	63.9	**3.3**	**64.9**	4.0	**3.3**	**64.9**
V	20	5.8	68.3	5.8	68.7	4.2	70.7	4.2	70.8	5.8	68.5	4.0	4.2	69.6
	40	8.0	74.2	8.0	73.7	6.1	76.6	6.1	76.7	8.9	72.9	7.0	8.9	72.9
	60	7.6	75.6	6.9	76.9	5.6	78.6	6.4	78.3	8.9	75.7	6.0	6.9	77.0
	80	7.1	76.3	7.6	76.7	5.8	79.6	6.2	78.9	8.0	76.9	7.0	6.7	77.9
	100	6.6	80.9	8.1	80.4	5.3	84.3	6.5	82.8	8.1	79.7	7.0	6.9	80.6
	All	7.0	75.1	7.3	75.3	**5.4**	**78.0**	5.9	77.5	7.9	74.7	6.2	6.7	75.6
VI	20	0.0	29.2	0.0	29.2	0.0	29.2	0.0	29.2	0.0	29.2	0.0	0.0	29.2
	40	40.0	49.1	40.0	49.1	40.0	47.4	40.0	47.4	30.0	50.3	40.0	40.0	47.6
	60	5.0	66.2	5.0	66.0	0.0	70.0	0.0	70.2	5.0	66.2	5.0	5.0	65.9
	80	0.0	66.7	0.0	67.0	0.0	68.7	0.0	67.9	0.0	68.0	0.0	0.0	68.6
	100	10.0	72.6	10.0	72.4	6.7	76.0	6.7	75.4	6.7	75.7	7.0	6.7	75.2
	All	11.0	56.4	11.0	56.4	9.3	**58.6**	9.3	58.4	**8.3**	57.9	10.4	10.3	57.3
VII	20	13.0	69.2	13.0	69.0	11.0	71.9	11.0	72.1	13.0	68.6	11.0	13.0	68.2
	40	9.3	77.2	11.6	75.7	7.3	80.7	7.3	80.5	8.3	79.0	8.0	8.3	78.5
	60	8.5	80.5	8.5	81.1	4.9	85.3	6.3	83.9	7.0	82.6	6.0	6.3	83.0

	80	10.2	81.2	9.1	82.4	7.6	85.1	8.6	83.7	9.6	81.2	10.0	9.6	81.0
	100	9.6	82.3	9.2	82.4	6.6	86.4	6.6	86.2	8.4	84.4	8.0	8.4	84.2
	All	10.1	78.1	10.3	78.1	**7.5**	**81.9**	8.0	81.3	9.3	79.2	8.6	9.1	79.0
VIII	20	12.0	69.5	12.0	69.1	10.0	72.3	10.0	72.3	10.0	72.1	10.0	10.0	72.2
	40	11.4	76.6	9.3	78.3	9.3	80.3	9.3	79.7	13.3	76.0	10.0	13.3	74.9
	60	9.9	80.1	9.9	79.7	6.3	84.9	6.3	84.6	7.0	83.2	7.0	7.0	83.5
	80	9.1	81.4	9.2	80.7	7.1	85.3	8.1	83.7	8.6	82.6	8.0	10.1	80.9
	100	8.3	82.5	7.9	82.6	5.4	87.3	5.9	86.0	8.3	84.7	9.0	8.3	84.7
	All	10.2	78.0	9.7	78.1	**7.6**	**82.0**	7.9	81.3	9.4	79.7	8.8	9.7	79.2
IX	20	0.7	43.0	0.7	43.0	0.0	43.6	0.0	43.6	0.0	43.6	0.0	0.0	43.6
	40	1.1	45.6	1.1	45.6	1.1	45.8	1.1	45.7	1.1	45.7	0.1	1.1	45.7
	60	0.7	43.5	0.7	43.5	0.7	43.6	0.7	43.6	0.7	43.6	0.1	0.7	43.6
	80	0.9	45.0	0.9	45.0	0.9	45.1	0.9	45.1	0.9	45.0	0.1	0.9	45.0
	100	0.7	46.0	0.7	46.0	0.7	46.1	0.7	46.1	0.7	46.0	0.1	0.7	46.1
	All	0.8	44.6	0.8	44.6	**0.7**	**44.8**	**0.7**	44.8	**0.7**	44.8	0.8	**0.7**	44.8
X	20	12.5	67.4	12.5	66.9	12.5	68.5	12.5	68.5	15.0	66.3	12.0	13.0	68.5
	40	6.1	78.4	6.1	78.0	6.1	79.9	6.1	79.6	6.1	79.2	6.0	6.1	79.6
	60	9.7	80.2	8.5	80.6	6.7	84.5	7.6	83.4	5.8	85.2	6.0	5.8	85.3
	80	7.4	82.8	6.4	83.2	5.6	85.9	5.6	85.6	4.8	89.0	5.0	4.8	89.2
	100	7.3	83.0	6.6	83.5	4.0	88.0	4.0	87.1	4.1	86.9	5.0	4.8	86.1
	All	8.6	78.3	8.0	78.4	7.0	81.4	7.2	80.9	7.2	81.3	**6.8**	6.9	**81.7**
Overall		7.3	69.9	7.2	70.0	**5.3**	**72.6**	5.6	72.2	6.0	71.5	6.0	6.2	71.3

Table 4
Values of average RPD:B, MSU and RPD:L for local search algorithms for 2DBPP with DD: solutions with smallest L_{max}.

Due date	Data	SGA			MXGA			UTS_{BFB}			RDM		
class	class	RPD:B	MSU	RPD:L	RPD:B	MSU	RPD:L	RPD:B	MSU	RPD:L	RPD:B	MSU	RPD:L
A	I	5.4	83.1	16.6	4.1	85.3	12.4	5.0	83.4	16.0	8.7	78.8	22.2
	II	3.3	63.7	17.4	2.0	66.2	11.1	2.5	64.9	13.2	2.5	65.4	12.0
	III	10.9	71.4	30.9	7.7	75.4	22.0	8.2	74.5	27.9	9.0	73.2	26.8
	IV	4.7	60.7	21.7	4.7	61.7	17.1	3.3	62.3	19.1	4.0	61.8	18.9
	V	8.7	72.5	24.2	7.0	74.5	17.9	7.7	73.6	22.0	7.4	73.5	21.7
	VI	11.0	54.5	23.2	9.3	56.1	16.6	11.0	54.4	21.5	10.3	55.3	19.5
	VII	11.8	74.5	33.5	8.9	78.5	23.5	10.7	76.7	29.7	9.0	77.1	29.4
	VIII	12.1	74.2	33.9	8.8	78.8	23.3	10.1	77.3	30.0	10.1	76.3	29.0
	IX	0.7	44.1	1.7	0.7	44.1	1.7	0.7	42.9	1.7	0.7	43.2	2.0
	X	9.7	75.0	27.9	7.9	77.3	23.8	7.9	76.6	32.0	9.0	74.9	27.5
	Average	7.8	67.4	23.1	**6.1**	**69.8**	**16.9**	6.7	68.7	21.3	7.1	67.9	20.9
B	I	6.4	81.8	34.9	4.6	84.7	24.2	6.8	81.7	31.7	8.8	78.4	38.2
	II	3.3	63.6	47.7	2.7	65.6	34.0	3.7	64.1	39.7	3.2	63.9	33.4
	III	12.3	68.9	66.8	8.7	73.9	46.2	12.7	70.0	65.0	10.7	71.6	56.4
	IV	6.0	59.3	53.4	4.7	61.7	36.0	6.3	59.6	49.1	6.0	59.2	45.7
	V	11.0	69.7	48.6	8.1	73.4	35.5	10.4	70.9	48.3	9.4	71.6	40.3
	VI	11.0	54.4	48.8	11.0	54.9	37.7	9.0	55.4	46.3	9.7	55.0	42.0
	VII	13.0	72.9	71.9	10.2	76.8	52.2	13.3	73.5	65.8	12.0	74.4	58.1
	VIII	14.1	72.2	72.7	9.0	77.4	49.4	11.2	75.1	67.3	11.7	74.3	60.5
	IX	0.7	43.9	2.4	0.7	44.0	2.4	0.7	43.1	2.5	0.7	43.4	3.6
	X	11.1	73.4	67.4	8.6	76.3	53.5	12.2	73.0	81.0	10.1	73.2	64.4
	Average	8.9	66.0	51.5	**6.8**	**68.9**	**37.1**	8.6	66.6	49.7	8.2	66.5	44.3
C	I	8.4	79.3	136.7	5.4	83.5	93.0	8.1	79.8	115.3	10.1	76.6	128.0
	II	5.0	61.8	232.1	4.0	64.0	149.5	4.8	62.5	165.3	4.0	62.5	179.6
	III	16.2	65.9	180.4	9.1	73.3	124.9	14.7	68.0	173.8	12.5	69.1	148.1
	IV	7.0	58.7	223.2	5.3	60.7	153.2	6.3	60.0	210.7	6.3	59.2	183.1
	V	13.2	67.3	149.2	8.8	72.4	105.0	13.2	68.2	142.1	10.4	70.0	121.0
	VI	11.0	54.4	274.9	11.0	54.5	241.2	11.0	54.3	264.4	11.7	53.9	251.4
	VII	14.6	70.2	296.5	10.5	76.2	209.6	16.3	70.3	262.0	13.4	71.8	227.3
	VIII	14.5	70.9	421.5	10.1	76.8	273.3	16.7	69.6	387.1	13.2	72.3	320.4
	IX	0.7	43.7	9.9	0.7	43.8	9.9	0.8	43.1	15.1	0.8	43.3	18.7
	X	12.3	71.3	396.6	9.0	75.3	318.5	12.5	70.8	412.6	13.1	71.0	345.2
	Average	10.3	64.4	232.1	**7.4**	**68.1**	**167.8**	10.5	64.7	214.8	9.6	65.0	192.3
Overall Average		9.0	65.9	102.2	**6.8**	**68.9**	**73.9**	8.6	66.7	95.3	8.3	66.5	85.8

7.3. Comparison of local search algorithms for 2DBPP

In this subsection, we provide computational results that compare the performance of our proposed MXGA with SGA, UTS_{BFB}, UTS_{TP} and RDM for the classical 2DBPP. Recall that SGA is identical to SXGA except that a single pair of offspring is produced from each pair of parents. We evaluate both 1-point and 2-point crossover for each of SGA and MXGA, respectively. Results for UTS_{TP} are those provided by Lodi et al. (1999b) and are based on a CPU time limit of 60 s per instance using a Silicon Graphics INDY R10000sc. However, computational results of Lee (2006) indicate that UTS_{BFB} and UTS_{TP} compete closely across all test instances and the overall average performance is the same. For a fair comparison between SGA, MXGA, UTS_{BFB}, and RDM in our tests, we employ a stopping criterion of 120 CPU seconds per instance. Apart from UTS_{TP}, all algorithms use BFB for the placement of rectangles.

Our computational results are presented in Table 3. For each algorithm, apart from UTS_{TP}, the entries in the first column report the average value of RPD:B, while the entries in the second column give the mean squared utilization of the bins (MSU), both averages being computed over the ten generated instances. The average by data class and the overall average value for all classes are also provided. It is clear that MXGA with the single point crossover exhibits better or at least equal performance relative to all of the other algorithms with respect to the average value of RPD:B as a performance measure for all but classes IV, VI and X. Similarly, with respect to the average value of MSU, MXGA exhibits superiority except for data classes II, IV and X. These results indicate a clear preference for the 1-point crossover over the 2-point crossover, and also for the multicrossover approach used in MXGA over the single crossover used in SGA. Further, the 1-point MXGA is superior to the two UTS algorithms and also to RDM.

A closer scrutiny of the results for UTS_{BFB}, UTS_{TP} and RDM show that they each exhibit reasonable performance, with the two versions of UTS performing marginally better. This supports the notion that the acceptance rule and randomization procedure introduced into RDM are comparable with the ideas of tabu lists and tabu tenure used within Unified Tabu Search.

7.4. Comparison of local search algorithms for 2DBPP with DD

Our final set of computational experiments compare the results of applying local search algorithms that are adapted for solving the 2DBPP with DD. As above, we again use $t=5$ in MXGA. Also, as suggested from the previous results, we employ the 1-point crossover operator in both MXGA and SGA.

Recall that for the 2DBPP with DD, our goal is to optimize the bicriteria objective function of the problem by alternating between minimizing the maximum lateness and minimizing the number of bins. In our implementation, the number of generations executed

Table 5
Values of average RPD:B, MSU and RPD:L for local search algorithms for 2DBPP with DD: solutions with smallest number of bins.

Due date	Data	SGA			MXGA			UTS_{BFB}			RDM		
Class	Class	RPD:B	MSU	RPD:L	RPD:B	MSU	RPD:L	RPD:B	MSU	RPD:L	RPD:B	MSU	RPD:L
A	I	3.8	86.0	22.2	3.5	86.2	13.9	3.7	85.6	18.8	5.1	79.1	22.5
	II	3.3	63.8	17.3	2.0	66.3	11.2	2.0	65.3	13.0	2.0	65.7	12.1
	III	7.9	74.9	38.3	6.8	76.4	23.6	5.8	77.3	30.2	6.8	74.8	29.0
	IV	4.7	60.8	21.7	4.7	61.7	17.2	3.3	62.3	19.0	4.0	61.8	18.8
	V	6.6	74.7	27.5	6.2	75.6	20.2	5.5	76.3	26.2	7.0	73.9	21.8
	VI	11.0	54.7	23.1	9.3	56.2	16.5	11.0	54.4	21.3	10.3	55.4	19.3
	VII	9.1	77.9	39.4	7.5	80.2	28.2	6.6	81.5	34.1	9.2	77.7	30.6
	VIII	8.8	78.3	39.8	7.9	80.2	28.3	7.1	80.8	33.3	9.5	77.2	29.3
	IX	0.7	44.2	1.7	0.7	44.2	1.5	0.7	42.9	1.6	0.7	43.2	2.0
	X	7.7	77.2	30.0	7.1	78.3	26.4	6.1	79.8	35.1	8.1	76.0	28.3
	Average	6.4	69.2	26.11	5.6	70.5	**18.7**	**5.2**	**70.6**	23.3	6.3	68.5	21.4
B	I	4.0	85.3	46.1	3.4	86.3	28.7	3.4	86.0	39.8	5.8	79.4	42.4
	II	3.3	63.6	47.7	2.0	66.2	34.1	2.0	65.5	40.2	2.0	64.8	33.4
	III	7.6	74.9	89.8	7.1	76.0	55.0	6.1	77.1	76.4	7.4	74.9	68.2
	IV	5.3	59.7	53.6	4.7	61.9	35.9	4.3	61.2	49.4	4.3	60.6	45.5
	V	6.7	74.5	63.5	6.0	75.7	40.7	5.6	76.2	56.0	7.4	73.7	48.0
	VI	11.0	54.4	48.8	10.3	55.5	37.7	8.3	55.8	46.5	9.0	55.5	42.0
	VII	8.8	77.9	91.0	7.8	79.5	66.7	6.5	81.4	82.0	9.2	77.4	65.8
	VIII	8.8	78.3	89.9	7.8	79.7	59.4	6.8	81.4	82.1	9.5	76.6	68.2
	IX	0.7	43.9	2.4	0.7	44.0	2.4	0.7	43.1	2.4	0.7	43.4	3.6
	X	7.8	77.5	75.0	7.1	78.2	60.0	7.0	78.4	92.6	7.8	76.4	73.1
	Average	6.4	69.0	60.80	5.7	70.3	**42.1**	**5.1**	**70.6**	56.7	6.3	68.3	49.0
C	I	4.1	85.3	189.4	3.4	86.0	124.4	3.5	85.7	149.0	5.9	78.2	138.1
	II	3.7	63.0	234.6	2.0	65.7	152.7	2.0	65.0	168.5	2.0	64.3	181.7
	III	7.4	75.3	253.5	7.0	75.7	147.6	5.2	78.1	227.0	8.1	73.8	182.3
	IV	5.3	60.0	225.3	4.7	61.2	154.2	4.0	61.8	213.2	4.5	60.7	186.7
	V	7.2	74.0	200.5	6.0	75.6	145.1	5.4	76.2	184.5	7.7	72.6	146.3
	VI	11.0	54.4	274.7	11.0	54.4	241.1	10.3	55.2	264.4	10.3	54.8	252.2
	VII	8.7	78.2	386.5	7.8	79.4	263.0	6.5	81.3	332.0	10.7	74.9	267.4
	VIII	8.9	77.9	509.1	8.1	79.4	328.6	6.8	81.0	467.6	9.8	75.8	450.7
	IX	0.7	43.7	9.7	0.7	43.8	9.9	0.7	43.3	16.4	0.8	43.4	21.6
	X	7.2	77.7	497.1	7.0	78.3	418.8	6.5	79.0	487.2	8.2	76.3	441.0
	Average	6.4	69.0	278.0	5.8	70.0	**198.5**	**5.1**	**70.7**	251.0	6.8	67.5	226.8
Overall average		6.4	69.1	121.6	5.7	70.3	**86.4**	**5.1**	**70.6**	110.3	6.5	68.1	99.1

before alternating the objective function is set using $G=100$, and the number of iterations executed for both UTS_{BFB} and RDM before alternating the objective function is also 100. By alternating the objective functions during the execution of the algorithms, we are solving the problem using a simultaneous optimization approach. Under this approach, both objective functions are treated as equally important. As a result, we approximate the set of *Pareto optimal* solutions corresponding to our two objective functions. Note that there are likely to be only a small number of Pareto solutions because the number of bins used in any "reasonable" solution is likely to exhibit little variability.

We present only the results for the two extreme points of our approximation of the Pareto optimal solutions, specifically those solutions with the minimum value of the maximum lateness and with the minimum number of bins. The computational results for these pairs of solutions are presented in Tables 4 and 5. In these tables, the first two columns give the due date class and the data class. The average values of RPD:B, MSU and RPD:L are listed for each of MXGA, MGA, UTS_{BFB} and RDM. For each due date class, the final line gives the average value over all data classes. Further, the final line of each table gives the overall average value over all due date and data classes.

From Table 4 where the solutions with the smallest value of L_{max} are summarized, it is clear that MXGA produces better quality solutions compared to other algorithms. The performance of UTS_{BFB} and RDM is similar, while SGA yields inferior solutions on an average. Although the average values of RPD:B are larger than those in Table 3, the differences are not large. This suggests that solutions with a small L_{max} also use a relatively small number of bins, and therefore the two objective functions do not always conflict.

The pattern of results in Tables 5 that summarize the solutions with the smallest number of bins exhibit a more varied pattern. The best solutions in terms of numbers of bins are provided by UTS_{BFB} with MXGA coming second best. However, for these solutions, the ones generated by MXGA have the smaller values of L_{max} on average. Also, the solutions for UTS_{BFB} that are able to pack the rectangles using a small number of bins come at the expense of relatively large L_{max} values. This outcome is not surprising as the original UTS_{BFB} is designed specifically for the classic 2DBPP where the sole objective is to minimize the number of bins used. It is interesting in Table 5 that the average value of RPD:B obtained from SGA of 6.4 and from UTS_{BFB} of 5.1 are generally better than the those obtained from the classic 2DBPP (7.4 for SGA and 6.0 for UTS_{BFB} in Table 3). However, MXGA and RDM fail to improve the solution quality in terms of RPD:B when compared with their counterparts in the classic problem.

We might expect the problem instances in due date class C to be the most challenging since the rectangle due dates are spread throughout the time horizon. The results for the average RPD:L in Tables 4 and 5 do not contradict this, although it is difficult to draw firm conclusions because the quality of the lower bound on L_{max} that is given in Eq. (3) affects the values of RPD:L.

8. Concluding remarks

In this paper, a new two-dimensional rectangular, single bin size, bin packing problem is defined where the rectangles to be packed have associated due dates. The objective is twofold: to minimize the number of bins used, and to minimize the maximum lateness. This problem helps to build bridges between the fields of bin packing and production scheduling.

A new heuristic placement routine for two-dimensional bin packing called Best Fit Bin (BFB) is described, and its performance is compared with the best performing placement heuristics in the bin packing literature. When considering both its computational complexity and solution quality, BFB is an attractive choice.

A multicrossover genetic algorithm (MXGA) is proposed to solve the classical non-oriented 2DBPP, the new problem variant with due dates and also the bicriteria problem. Various devices have been introduced into the MXGA to further enhance the solutions generated. In comparative computational results for the classical 2DBPP, MXGA achieves better performance compared to a single crossover genetic algorithm, the Unified Tabu Search (UTS) method of Lodi et al. (1999a, 1999b, 2004), and to a randomized descent method. However, the relative quality of the results is not so clear cut for the 2DBPP with DD, where the MXGA has mixed success when compared with UTS.

References

Arbib, C., Marinelli, F., Pezzella, F., 2012. An LP-based tabu search for batch scheduling in a cutting process with finite buffers. International Journal of Production Economics 136, 287–296.

Baker, B.S., Coffman Jr., E.G., Rivest, R.L., 1980. Orthogonal packing in two dimensions. SIAM Journal on Computing 9, 846–855.

Beasley, J.E., 2004. A population heuristic for constrained two-dimensional non-guillotine cutting. European Journal of Operations Research 156, 601–627.

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

Burke, E.K., Kendall, G., Whitwell, G., 2004. A new placement heuristic for the orthogonal stock-cutting problem. Operations Research 52, 655–671.

Burke, E.K., Kendall, G., Whitwell, G., 2009. A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock cutting problem. INFORMS Journal of Computing 21, 505–516.

Chazelle, B., 1983. The bottom-left bin packing heuristic: an efficient implementation. IEEE Transactions on Computers 32, 697–707.

Coffman Jr., E.G., Garey, M.R., Johnson, D.S., 1984. Approximation algorithms for bin-packing—an updated survey. In: Ausiello, G., Lucertini, N., Serafini, P. (Eds.), Algorithm Design for Computer Systems Design. Springer-Verlag, New York, pp. 49–106.

Dowsland, K.A., Dowsland, W.B., 1992. Packing problems. European Journal of Operations Research 56, 2–14.

Dell'Amico, M., Martello, S., Vigo, D., 2002. A lower bound for the non-oriented two-dimensional bin packing problem. Discrete Applied Mathematics 118, 13–24.

Dyckhoff, H., Finke, U., 1992. Cutting and Packing in Production and Distribution. Physica Verlag, Heidelberg, Germany.

Esquivel, S.C., Leiva, A., Gallard, R.H., 1997. Multiple crossover per couple in genetic algorithms. In: IEEE International Conference on Evolutionary Computation, pp. 103–106.

Evans, G.W., 1984. An overview of techniques for solving multiobjective mathematical programs. Management Science 30, 1268–1282.

Falkenauer, E., Delchambre, A., 1992. A genetic algorithm for bin packing and line balancing. In: Proceedings of the 1992 IEEE International Conference on Robotics and Automation. IEEE Computer Society Press, Los Alamitos, CA, pp. 1186–1192.

Garey, M.R., Johnson, D.S., 1979. Computer and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, San Francisco.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Kluwer Academic Publishers, Boston, MA.

Gonçalves, J.F., Resende, M.G., 2011. A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. Journal of Combinatorial Optimisation 22, 180–201.

Gramani, M.C.N., França, P.M., 2006. The combined cutting stock and lot sizing problem in industrial processes. European Journal of Operations Research 174, 509–521.

Hendry, L.C., Fok, K.K., Shek, K.W., 1996. A cutting stock and scheduling problem in the copper industry. Journal of the Operational Research Society 47, 38–47.

Herrera, F., Lozano, M., Pérez, E., Sánchez, A.M., Villar, P., 2002. Multiple crossover per couple with selection of the two best offspring: an experimental study with the BLX-α crossover for real-coded genetic algorithms. In: Advances in Artificial Intelligence—IBERAMIA 2002. Lecture Notes in Computer Science, vol. 2727. Springer, Berlin. pp. 392–401.

Hopper, E., Turton, B.C.H., 1999. A genetic algorithm for a 2D industrial packing problem. Computers and Industrial Engineering 37, 375–378.

Hopper, E., Turton, B.C.H., 2001a. A review of the application of meta-heuristic algorithms to 2D strip packing problems. Artificial Intelligence Review 16, 257–300.

Hopper, E., Turton, B.C.H., 2001b. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European Journal of Operations Research 128, 34–57.

Hwang, S.-M., Kao, C.-Y., Horng, J.-T., 1994. On Solving Rectangle Bin Packing Problems using Genetic Algorithms. In: Proceedings of the 1994 IEEE International Conference on Systems, Man, and Cybernetics, vol. 2. IEEE, San Antonio, TX, pp. 1583–1590.

Jakobs, S., 1996. On genetic algorithms for the packing of polygons. European Journal of Operations Research 88, 165–181.

Kroger, B., 1995. Guillotineable bin packing—a genetic approach. European Journal of Operations Research 84, 645–661.

Lee, L.S., 2006. Multicrossover Genetic Algorithms for Combinatorial Optimisation Problems. PhD Thesis. School of Mathematics, University of Southampton, UK.

Leung, T.W., Yung, C.H., Troutt, M.D., 2001. Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem. European Journal of Operational Research 145, 530–542.

Leung, T.W., Yung, C.H., Troutt, M.D., 2003. Applications of a mixed simulated annealing-genetic algorithm for the two-dimensional orthogonal packing problem. Computers and Industrial Engineering 40, 201–214.

Li, S., 1996. Multi-Job cutting stock problem with due dates and release dates. Journal of the Operational Research Society 47, 490–510.

Liu, D., Teng, H., 1999. An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. European Journal of Operations Research 112, 413–420.

Lodi, A., Martello, S., Vigo, D., 1999a. Approximation algorithms for the oriented two-dimensional bin packing problem. European Journal of Operations Research 112, 158–166.

Lodi, A., Martello, S., Vigo, D., 1999b. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. INFORMS Journal of Computing 11, 345–357.

Lodi, A., Martello, S., Vigo, D., 2002a. Recent advances on two-dimensional bin packing problems. Discrete Applied Mathematics 123, 379–396.

Lodi, A., Martello, S., Vigo, D., 2002b. Two-dimensional packing problems: a survey. European Journal of Operations Research 141, 241–252.

Lodi, A., Martello, S., Vigo, D., 2004. TSpack: a unified tabu search code for multi-dimensional bin packing problems. Annals of Operations Research 131, 203–213.

Martello, S., Vigo, D., 1998. Exact solution of the two-dimensional finite bin packing problem. Management Science 44, 388–399.

Nonas, S.L., Thorstenson, A., 2000. A combined cutting-stock and lot-sizing problem. European Journal of Operations Research 120, 327–342.

Puchinger, J., Raidl, G.R., Koller, G., 2004. Solving a real-world glass cutting problem. In: EvoCOP, vol. 3004, pp. 165–176.

Reinertsen, H., Vossen, T.W.M., 2010. The one-dimensional cutting stock problem with due dates. European Journal of Operations Research 210, 701–711.

Smith, D., 1985. Bin-packing with adaptive search. In: Grefenstette, J.J. (Ed.), Proceedings of the 1st International Conference on Genetic Algorithms and their Applications. Lawrence Erlbaum Associates, Hillsdale, NJ, pp. 202–206.

Wäscher, G., Haußner, H., Schumann, H., 2007. An improved typology of cutting and packing problems. European Journal of Operations Research 183, 1109–1130.

Zheng, W., Ren, P., Ge, P., Qiu, Y., Liu, Z., 2012. Hybrid heuristic algorithm for two-dimensional steel coil cutting problem. Computers and Industrial Engineering 62, 829–838.