European Journal of Operational Research 209 (2011) 219–227

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A hybrid placement strategy for the three-dimensional strip packing problem

S.D. Allen *, E.K. Burke, G. Kendall

University of Nottingham, School of Computer Science, Jubilee Campus, Nottingham NG8 1BB, UK

ARTICLE INFO

Article history:
Received 19 November 2009
Accepted 18 September 2010
Available online 24 September 2010

Keywords:
Cutting/packing
Heuristics
Metaheuristics

ABSTRACT

This paper presents a hybrid placement strategy for the three-dimensional strip packing problem which involves packing a set of cuboids ('boxes') into a three-dimensional bin (parallelepiped) of fixed width and height but unconstrained length (the 'container'). The goal is to pack all of the boxes into the container, minimising its resulting length. This problem has potential industry application in stock cutting (wood, polystyrene, etc. – minimising wastage) and also cargo loading, as well as other applications in areas such as multi-dimensional resource scheduling. In addition to the proposed strategy a number of test results on available literature benchmark problems are presented and analysed. The results of empirical testing of the algorithm show that it out-performs other methods from the literature, consistently in terms of speed and solution quality-producing 28 best known results from 35 test cases.

1. Introduction

Although cutting and packing research has received significant attention in recent years it has generally concentrated on one- or two-dimensional packing. Though this has many real-life practical applications – paper, metal, glass or other sheet material cutting, basic pallet loading, etc., and several more theoretical ideas such as multi-processor scheduling (Coffman et al., 1978) it is not directly applicable or translatable to other industrial problems, such as three-dimensional block cutting (e.g. foam, wood, marble, etc.), load planning (trucks, air cargo, etc.) or multi-dimensional limited resource scheduling. In recent years, the reliance on automation has increased, leading to a requirement of fast, efficient cutting and packing algorithms in order to semi-automate cutting and packing procedures, and in some cases (such as Computer Numerical Control machines) fully automate these procedures. It has been suggested that automated approaches can be as efficient, or in some cases more efficient, than a human attempting the same problem (Roberts, 1984) and usually in a substantially shorter amount of time (Hower et al., 1996).

Three-dimensional strip packing can be defined as the following: given a three-dimensional bin (parallelepiped) of fixed width and height but unconstrained length (the 'container'), and a list of smaller cuboids ('boxes') of given width, length and height, the goal is to pack the boxes into the container, such that all vertices of each box lie within the bounds of the container, minimising the container's resulting length. An obvious constraint is that the packed boxes must not overlap (i.e. their interiors are disjoint). Also the boxes must be packed feasibly – i.e. they must be packed orthogonally (parallel to the axes of the container). A visual example of a strip packing container can be seen in Fig. 1. According to the cutting and packing typology of Wäscher et al. (2007) this problem is classified as the 3D regular open-dimension-problem (ODP) with one open dimension.

Even though two-dimensional strip packing and its variants have been studied since the 1960s (for reviews of metaheuristic approaches to the two-dimensional strip packing problem, see Hopper and Turton (2001a), Riff et al. (2009)) the first approximation of the three-dimensional strip packing problem was not published until 1990 by Li and Cheng who released an improved version of the algorithm two years later (Li and Cheng, 1992). These papers along with others (e.g. Miyazawa and Wakabayashi, 2004; Jansen and Solis-Oba, 2006; Bansal et al., 2007) focus on algorithmic performance analysis, though there are some with actual implementation details and empirical results; more recently Karabulut and Inceoglu (2004), Bortfeldt and Gehring (1999), Bortfeldt and Mack (2007). The three-dimensional strip packing problem is NP-hard due to it being a generalisation of the uni-dimensional packing problem. The uni-dimensional packing problem, otherwise known as the classical 'bin packing problem,' involves placing a number of one-dimensional objects with given sizes into a number of one-dimensional 'bins,' or containers, of fixed capacity. The objective is to minimise the number of bins used. The NP-hardness of bin packing (as a reduction from PARTITION) is given in Garey and Johnson (1979). Due to the NP-hardness of the one-dimensional problem, and subsequently of the three-dimensional version, (meta-)heuristic approaches are generally required to obtain feasible solutions to large instances in a realistic amount of time.

The packing strategy presented here draws inspiration from the ideas proposed for the equivalent two-dimensional problem

* Corresponding author. Tel.: +44 (0) 115 84 66520.
E-mail address: sda@cs.nott.ac.uk (S.D. Allen).

0377-2217/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2010.09.023

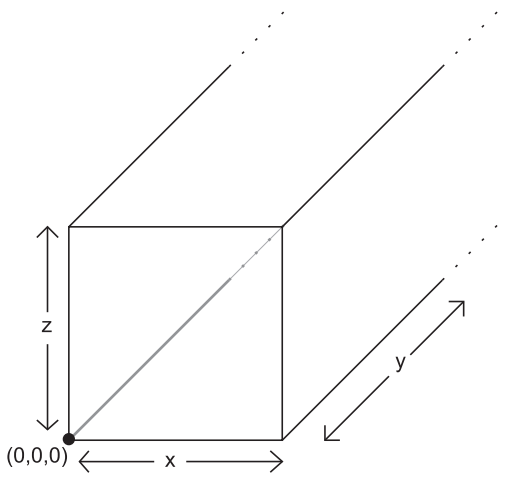

Fig. 1. A visual representation of a three-dimensional strip packing container.

(Burke et al., 2004; Burke et al., 2009), whilst adding a number of substantial changes and improvements in order to make it applicable to the three-dimensional problem.

2. The 3BF placement heuristic

This section introduces a new heuristic algorithm, which we refer to as the three-dimensional best fit (3BF) for the three-dimensional strip packing problem. We also present an implementation for the algorithm in order to gauge the results against a number of available benchmark tests.

In many heuristic algorithms designed for the two-dimensional problem, the solution is found by iteratively taking the head of the list of input rectangles (usually sorted by a common property such as width or volume). The algorithm on which 3BF draws some of its inspiration from is Burke et al. (2004), which we refer to as 2BF. Though inspiration is drawn from this paper and the general methodology (i.e. a 'best fit' scheme) is the same, there are many significant differences which are discussed in later sections.

Instead of searching the list sequentially, the list is searched dynamically in order to find the most suitable rectangles to fill the available gaps. We define gaps as suitable locations at the furthest points of a packing in which a box may be placed, unobstructed by other boxes. This means that gaps are effectively two-dimensional 'sites' for a box to be positioned in. This is referred to as a 'skyline' representation in the paper by Burke et al., i.e. the two-dimensional skyline in a three-dimensional packing can be imagined as a heightmap with only rectilinear features (the height/shade is defined by the furthest face of the furthest box in the packing at any point), where every rectangle that can be drawn enclosing a single shade represents a gap that a box can be placed into. In the same paper by Burke et al. the methodology of greedily placing boxes based on their size is referred to as 'best fit'. This method is also employed by our proposed 3BF methodology, which attempts to find a box whose base/footprint (we use the term 'footprint' of a box to refer to the deepest face which will be placed within a gap, i.e. the back of the box when rotated to any given orientation) fills the deepest available gap entirely or, failing that, to fill as much of the gap as possible-effectively making this a greedy algorithm. When multiple orientations of a box or different boxes fill the same amount of the gap (or multiple gaps are available) then tie breaking is done according to additional rules (explained in Section 2.1), and remaining ties are broken arbitrarily.

As with the 2BF algorithm there are some issues with time complexity as, at each stage, first a gap needs to be found by searching the container profile and then a box has to be chosen from the input list. There are some attempts to relieve these time complexity bottlenecks as described in the later sections of this paper.

The representation of gaps and boxes used here is the same as the "extreme-point" method as introduced in Crainic et al. (2008). Initially, in an empty container, a single point will be available for boxes to be placed in – $(0,0,0)$. When a box is added to this location the point is removed and a number of new points are added to the list, allowing subsequent boxes to be placed on top of and around the newly placed box. For every box that is placed, nine new points are pushed back into the gap list (sorted by increasing y location, followed then by increasing z and x locations. The location of the new points are as in the paper of Crainic et al., which also proves the complexity of the point list update procedure as $O(n)$ where n is the number of boxes already placed in the container.

This representation, using a sorted point list, means that in order to find the best gaps in the box the list must only be searched from the beginning until a point with a higher y location than the first point in the list is reached in the case of the Neighbour Score and Maximum Contact placement strategies, or only the first point in the case of the Deepest-bottom-leftmost and Smallest Extrusion strategies (see Section 2.1). Once potential locations are found, suitable boxes are placed at each location in order to gauge their suitability and check for collisions with previously placed boxes (as explained in Section 3.2).

In order to speed up the input list dynamic search each input box is first oriented so that width $\geqslant$ length $\geqslant$ height. The list is then sorted into order of decreasing width (if widths are equal the second priority is height and the third is length). This means that only around $\frac{n}{2}$ inspections are needed at each iteration of the algorithm on average, as given a gap of dimensions i by j (where $i \geqslant j$) we need only traverse the input list until all boxes of dimensions equal to i and j is found (or the first closest answers), as any box found after this will be worse at fitting the gap than these boxes have already been.

As well as speeding up the search this also means that larger boxes are generally placed earlier on in the packing, which leads to a higher quality solution as there are fewer boxes protruding over the top of the three-dimensional skyline of the container.

After a gap has been selected, the list of input boxes is searched in order to find the best fit for the gap. The result will always be one of three options:

1. One or more boxes are found to have a rotation whose footprint has exactly the right dimensions to totally fill the gap, possibly after rotation. In this case, the box with the highest evaluation score (as explained in Section 2.1) is chosen.
2. One or more boxes are found to fit the gap, possibly after rotation, but without filling the gap entirely. If this is the case then the box which fills most of the gap (or in the case of multiple boxes filling the same amount of space, the box with the highest evaluation score) is chosen.
3. No boxes are found to fit the gap.

In the case of option 1, there is no choice as to where the box should be placed within the gap, so it only undergoes the rotations necessary to make it fit and it is placed. In the case of option 2 then the box that fills the most of the gap is used after any necessary rotations, with tie breaking and placement according to the placement strategy (see Section 2.1) currently in use. Finally, for option 3 the gap can successfully be ignored on future iterations of the algorithm (i.e. removed), as the input list of boxes is static.

A pseudocode representation of the 3BF algorithm is provided in Algorithm 1. The set D represents the set of box placements, where a placement consists of a box (originally from B), a specified orientation and a placement point as provided by the

getLowestGaps method. The *getLowestGaps* method returns a list of placement points (each representing a position along the *x*, *y*, and *z*-axis) that a box may be placed in – i.e. so that its lower back left vertex is positioned at the given point, taking into account the previously placed boxes. It is assumed that *getLowestGaps* marks gaps that are unusable (i.e. lie within the location of an already placed box in D) as invalid and ignores invalid gaps. Finally, *evaluationScore* is described in Section 2.1. We assume that all orientations, o, of any given box are valid.

Algorithm 1. Best-fit Packing Algorithm

Input: a set of boxes to place, B and a container, c
Output: a set of boxes with associated orientation and placement information, D

$D \leftarrow \emptyset$
while $|B| > 0$ **do**
  $G \leftarrow getLowestGaps(c, D)$
  $bestScore \leftarrow 0$
  $bestOrientation \leftarrow 0$
  for all i such that $1 \leqslant i \leqslant |G|$ **do**
    for all j such that $1 \leqslant j \leqslant |B|$ **do**
      for all orientations o such that $1 \leqslant o \leqslant 6$ **do**
        $t \leftarrow evaluationScore(g_i, b_j^o)$
        if $t > bestScore$ **then**
          $bestScore \leftarrow t$
          $bestBox \leftarrow j$, $bestGap \leftarrow i$, $bestOrientation \leftarrow o$
        end if
      end for
    end for
  end for
  if $bestScore \neq 0$ **then**
    position $b_{bestBox}^{bestOrientation}$ in $g_{bestGap}$
    remove $b_{bestBox}$ from B
    add placement of $b_{bestBox}^{bestOrientation}$ to D
  else
    for all i such that $1 \leqslant i \leqslant |G|$ **do**
      mark g_i as invalid for future iterations
    end for
  end if
end while
return D

2.1. Placement strategies

As with 2BF there are a number of placement strategies, although they are substantially different due to the third dimension having to be taken into consideration. There are also different priorities (i.e. the minimisation of the length of the filled container being the highest priority in 3BF, as opposed to the minimisation of the height of the packing in 2BF). The scores generated by the placement strategies are defined such that a higher score is deemed a better location for that orientated box/gap pair. The placement strategies are illustrated in (the two-dimensional) Fig. 2 and are as follows:

1. **Deepest-bottom-leftmost**: The box is placed at the deepest-bottom-leftmost position (i.e. with the lowest y position, ties broken by lowest z position and finally lowest x position) in the container, i.e. at the point closest to $(0,0,0)$. The evaluation function returns the area of the footprint of the box, with ties being broken by furthest extrusion in the y-axis, i.e. a bigger extrusion is deemed a better solution in this placement strategy. Large extrusions placed towards the end of a packing are dealt with by the tower processing technique described in Section 2.2.
2. **Maximum Contact**: The box with the largest volume is placed so that the sum of its surface areas in contact with other boxes and edges of the container is maximised. The faces of the box are weighted differently by multiplying the surface area (in contact with other boxes/the container) of the back by 4, the left side by 2, the underside by 2 and all other surfaces by 1. Obviously, changing the weightings can lead to different packings with different priorities, e.g. weighting the underside by more would lead to much more stable packings in practice.
3. **Smallest Extrusion**: As with the deepest-back-leftmost strategy, though the evaluation function returns 1 divided by the extrusion in the y-axis, thus the box with the smallest far point in the y direction (which fills the gap the most) is chosen and placed in the deepest-bottom-leftmost position.
4. **Neighbour Score**: The box with the largest footprint is placed so that it is touching as many boxes whose furthest face on the y-axis are equal to, or less than, that of the box being placed, i.e. neighbouring boxes that could form vertical walls for subsequent boxes to be placed in front of. This scoring method effectively takes all boxes surrounding the box being tested, and sums the perimeter which is in contact with any of those boxes. Boxes protruding further than the tested box/orientation pair are not included in the summation, and boxes with an identical protrusion to the tested box/orientation pair are given a weighting of 2, i.e. summed twice, as they directly create a larger gap for subsequent boxes to be placed in.

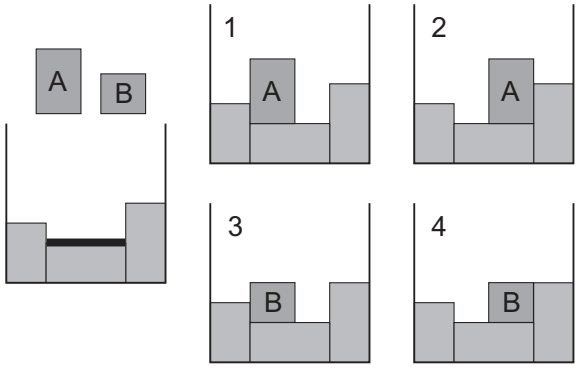

Fig. 2. The placement strategies in action. (Left) A partial packing and two non-rotatable candidate boxes (with identical widths) for packing next and the lowest gap indicated by a thicker line. (Right) Packing after one iteration following the (1) Deepest-bottom-leftmost, (2) Maximum Contact, (3) Smallest Extrusion and (4) Neighbour Score rules.

In the case of Fig. 2 it can be seen in part (1), the deepest-bottom-leftmost strategy packs box A into the position as it is the deepest, lowest position available and given that box A has an equal width but higher volume than box B it is packed. Part (3) follows the same logic but as box B has a lower volume it is packed instead. In part (2) of the diagram, box A is placed in the rightmost location of the gap as it shares more contact with the rightmost box previously packed within the container. Box A is chosen over B as it is earlier in the packing list. In part (4) of the diagram box B is placed in this location as it shares the same extrusion as the rightmost box and therefore creates a larger 'platform' for subsequent boxes to be placed on.

2.2. Tower processing

Due to the nature of the algorithm, solutions may potentially be found of relatively poor quality as 'towers' can be placed. These occur when boxes are placed late on in the packing process with their

rotations such that the greatest (or second greatest) dimension is assigned to the y-axis, causing them to stick out above the 'skyline' of the container. To combat this problem a further stage is introduced into the algorithm, where the tallest tower is taken and rotated so that the dimension currently assigned to the y-axis is assigned to either of the other two axes and a shorter dimension is assigned to the y-axis. It is then placed back somewhere within the solution in its new orientation. This is repeated until no improvement in solution quality can be found. A two-dimensional representation of this can be seen in Fig. 3 and the pseudocode is presented in Algorithm 2. The function *furthestPoint* returns the location (i.e. along the y-axis) of the front-most face, that is, the extent of the extrusion of the box in the skyline. The *getAllGaps* method works as the *getLowestGaps* method except that it returns all gaps that have not been marked as invalid rather than just those sharing the lowest y location.

Algorithm 2. Tower Processing Algorithm

Input: a set of boxes with orientation and position information, D, the container they have been placed in, c
Output: the set of placed boxes with orientation and position information, D, with 'tower' boxes processed where possible

loop
let $b_i^o \in D$ be a box with orientation o and position information, with maximum $furthestPoint(b_i^o)$
$j \leftarrow furthestPoint(b_i^o)$
$k \leftarrow j$
$b_i^{best} \leftarrow b_i^o$
remove b_i^o from D
for all orientations $p \in \{1,\ldots,6\}$ of b_i, b_i^p, such that $length(b_i^p) < length(b_i^o)$
  place b_i^p in lowest available gap from $getAllGaps(c,D)$
  if $furthestPoint(b_i^p) < j$ **then**
$j \leftarrow furthestPoint(b_i^p)$
$b_i^{best} \leftarrow b_i^p$
  end if
  remove b_i^p from D
end for
position b_i^{best} in lowest available gap from $getAllGaps(c,D)$
add placement information of $b^{bestOrientation}$ to D
$furthestPoint(b_i^{best}) = k$ **then**
  exit
end if
end loop
return D

3. Metaheuristic enhancements to 3BF

Though the solution results presented previously represent good quality in short computational times, the findings of Burke et al. (2009) suggest that with a hybrid heuristic/metaheuristic packing strategy the quality of solutions may be further improved. This leads to two distinct placement phases, which are shown in Fig. 5.

The aim of implementing a second, metaheuristic stage to the initial 3BF algorithm is to increase solution quality whilst maintaining short execution times. This effectively means creating a hybrid algorithm which will place the majority of the input list of boxes using the fast 3BF algorithm, and then placing the remaining boxes using a metaheuristic method for higher quality solutions.

The 3BF heuristic, coupled with a metaheuristic search method along with a decoding procedure (which we call the deepest-bottom-left-fill method, DBLF) gives the hybrid packing strategy. In order to produce orderings that may generate higher quality solutions the DBLF method is coupled with a metaheuristic-tabu search (see Section 3.3). Examples of metaheuristic approaches to two-dimensional cutting/packing can be seen in Lai and Chan (1997), Faina (1999)), and in three dimensions (Karabulut and Inceoglu, 2004; Mack et al., 2004).

3.1. Deepest-bottom-left-fill strategy

In the research area of two-dimensional cutting and packing problems the most commonly used method for packing regular and irregular shapes involves the bottom-left class of heuristics. These methods involve simply placing the input list of rectangles into the bottom-leftmost location on the packing sheet. Applications of this class appear in the literature as early as 1980 (Baker et al., 1980; Chazelle, 1983).

The main problem with using the bottom-left approach is that holes can appear in the packing sheet where subsequent rectangles are unable to be placed, as shown in Fig. 4.

The solution to this problem is suggested as the bottom-left-fill method, which maintains a list of potentially usable gaps (ordered by increasing distance from (0,0,0) with lowest depth being highest priority and furthest left being lowest priority) for boxes to be packed within and also a list of the previously placed boxes in order to check for collisions with subsequently placed boxes. This method has been shown to improve solution quality over pure bottom-left methods (Hopper and Turton, 1999). DBLF is an obvious extension of BLF as it simply places boxes in the deepest available position in three dimensions (i.e. closest to (0,0,0) – with the depth/ y-axis being of highest priority to minimise, the bottom/z-axis next and finally the left/x-axis) whilst filling available gaps where possible. This approach has also been used previously in literature (Karabulut and Inceoglu, 2004), where a detailed description can be found.

The DBLF method is an online algorithm (i.e. it processes each box in the input list in order, as and when the algorithm reaches it) and therefore the order of the input list is important as it determines the output solution. It has been shown in the literature that intelligently pre-ordered input sequences generally generate higher quality solutions (Coffman et al., 1984) and, specifically, ordering by decreasing dimensions tends to yield the best results (Hopper et al., 2001b).

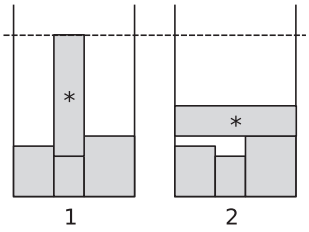

Fig. 3. A two-dimensional 'tower' (marked with an asterisk) being rotated in order to increase solution quality in a packing.

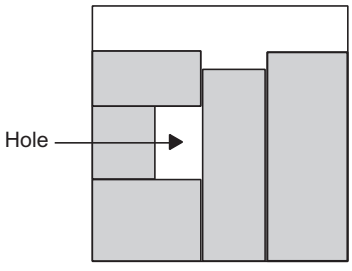

Fig. 4. 2D packing sheet with a hole.

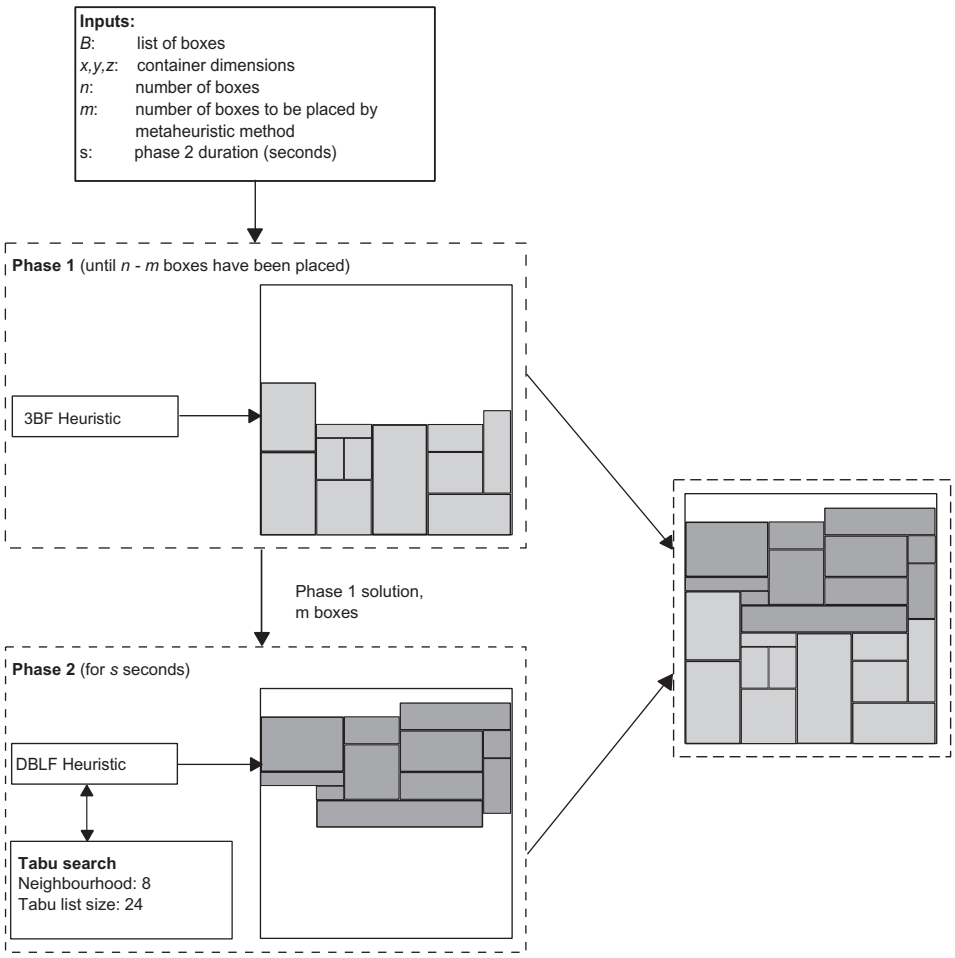

Fig. 5. A summary of the proposed process.

Again we refer the interested reader to Crainic et al. (2008) for specific details on this representation of boxes and gaps. Pseudocode for the DBLF algorithm is provided in Algorithm 3. The *getAllGaps* method returns all usable gaps in ascending order of their y positions (ties broken by ascending z then x positions). The *removeUnusableGaps* method removes all gap locations which lie within any placed boxes, i.e. marks them as invalid, meaning that no other boxes can be placed there without causing overlaps.

3.2. The DBLF method and axis-aligned bounding-box trees

If implemented naively, the collision detection in DBLF can take quadratic time, with n boxes requiring a total of $O(n^2)$ comparisons. To attempt to overcome this bottleneck a simple hierarchical space partitioning method used in many three-dimensional computer graphics and gaming systems (Walsh (2002) was implemented – the axis-aligned bounding box tree (AABB tree).

AABB trees are used for fast collision detection in some games and other three-dimensional modelling systems as they model each object in the 'world' with a 'bounding box' – effectively a cuboid that encompasses the entire object. Axis-aligned bounding boxes are a special case of bounding boxes which are always parallel to all three axes in three dimensions. Obviously as the objects being manipulated in this case are already cuboid in shape and can only perform rotations of 90° this is an ideal solution.

Firstly, the AABB tree partitions the container into smaller sections, by recursively splitting each section (initially the whole container) in half along its longest axis each time and placing each half into the child nodes of a binary tree structure, up to a maximum depth of 5 in this implementation. For every candidate box that is added to the AABB tree structure, the tree is searched recursively until the leaf space containing the location of the candidate box is found. Only boxes from this leaf are then individually tested for collisions with the candidate box, drastically cutting down the amount of time in the practical case for collision detection compared to the naive approach. It is worth noting, however, that the theoretical worst-case complexities of the two methods are the same.

Algorithm 3. DBLF Algorithm

Input: a set of ordered, unplaced boxes, B and a container, c
Output: a set of boxes with associated placement information, D

$D \leftarrow \emptyset$
for all i such that $1 \leqslant i \leqslant |B|$ **do**
  $G \leftarrow getAllGaps(c, D)$
  $placed \leftarrow false$
  for all j such that $1 \leqslant j \leqslant |G|$ **do**
    if $placed = false$ **and** $fits(b_i, g_j)$ **then**
    position b_i in g_j
    remove b from B
    add placement information of b to D
    $placed \leftarrow true$
    end if
  end for
end for
return D

3.3. Tabu search

The metaheuristic employed in manipulating the input list of m boxes to be assessed using DBLF is tabu search (see Glover and Laguna, 1993). This is effectively a hill climbing method employing the use of memory so that states in the search space are not re-explored – allowing the escape from local optima. Initially the input list for the search was a list of boxes rotated and sorted. Although it is suggested in Burke et al. (2009), Hopper et al. (2001b) that rectangles (or in this case boxes) are sorted by decreasing height, it was found through experimentation that boxes sorted by decreasing width as their first priority and decreasing height as the lowest priority created substantially better solutions for this problem. The tabu search method generates eight neighbours (i.e. direct moves from the current position) of the local state at random and assesses the fitness of each state by decoding it using the DBLF method as described previously. The method used to generate a neighbouring solution is straightforward. Two boxes from the given list are chosen at random and swapped by position. The first chosen box is then randomly rotated to one of its six possible orientations, each with equal probability ((1/6)th if all orientations are allowed) and this list is then returned to the DBLF function for evaluation. The neighbour with the highest score (meaning lowest overall solution height) is accepted as the next move and is added to the tabu list, meaning that it cannot be revisited again (though due to the finite nature of memory, and the linearly increasing time factor involved with checking if states are already in the tabu list, the length of the tabu list is limited to a certain number of states, s, and when s is reached the first state in the list is simply 'forgotten' and may be revisited). The parameters for the tabu search were decided upon after rigorous initial testing. The tabu search tended to work well on the datasets provided with a tabu list size of 24.

3.4. Adaptive value of m

During initial testing it was found that a fixed value of m (the number of boxes to be packed using the metaheuristic method) between approximately 10 and 35 yielded the best results, though the implementation competed more strongly when the value of m was allowed to adjust dynamically throughout each run. In this implementation m was started at 10 and was increased by a value of 2 if a higher solution quality was not found within 10 seconds of the run, i.e. after 10 seconds of running time if a higher quality solution had been found then m would remain the same for the next 10 seconds, otherwise it would increase by 2.

4. Comparative evaluation of the packing strategy

4.1. Experimentation and results of 3BF

The implementation of the algorithm is in C++ using the Standard Template Library (STL) where appropriate and compiled using the Microsoft Visual C++ 8.0 compiler. A viewer application used for visualisation and inspection of solution quality was also written in C++, using the OpenGL library (Allen, 2010). In order to try and get the best result possible eventually all four of the placement strategies are tried and the best result returned as the final solution.

Several different sources of test data were used to evaluate the performance of 3BF. Firstly the KI test data sets used in Karabulut and Inceoglu (2004) were attempted in order for a direct comparison with a previously established packing strategy producing the previous state of the art results (Using genetic algorithms along with the DBLF method, GA + DBLF. Though this method is probabilistic, no indication of the number of runs or whether the results are best results or average results across the runs is given by the authors.). Note that there is a constraint that no box may be rotated for this dataset. Secondly the BR data sets (thpack1–thpack10) originally suggested in Bischoff and Ratcliff (1995) and available to download from the OR-Library (http://people.brunel.ac.uk/mastjjb/jeb/info.html) were adapted for use in the strip packing problem by ignoring the container length dimension but maintaining the rotation restrictions as suggested in the original paper, in order to compare against (Bortfeldt and Mack, 2007) which is also the current state of the art for some of these data sets (BR6–BR10). The method proposed in Bortfeldt and Mack (2007) is deterministic and therefore no repeat runs were necessary in the original paper with regards to solution qualities. The current state of the art results for sets BR1–BR5 are however held by a parellelised tabu search method described in Bortfeldt and Gehring (1999). This paper also uses several variants of probabilistic and deterministic methods to generate solutions but only a single run of each method is used to determine the 'winning' method. It is worth noting that the problem formulation used in the paper includes additional stability constraints where boxes must be supported by the boxes beneath them, i.e. lower in the z-axis. As with the aforementioned papers, only the first 10 instances from each dataset were used in the comparison. Finally, the two-dimensional datasets proposed in Burke et al. (2004) were used for comparison with the different placement strategies to those proposed here (comparing a three-dimensional packing strategy to a two-dimensional strategy is, in this implementation, simply a case of setting all height values in the two-dimensional datasets to a fixed, arbitrarily small number). The packing methods given in Burke et al. (2004) are deterministic. The number of boxes, n, is shown in the tables for comparison purposes, with the value for the BR1–BR10 datasets being the mean number of boxes per instance. All tests were carried out on an Intel PC with a core speed of 1.86 GHz using single-threaded code.

Volume utilisation (as a percentage) in the tables can be calculated by taking the optimal length for the packing divided by maximum extreme y position over all the boxes, where the optimal length is known for the C1P1–C3P5 and N1–N13 datasets, and is calculated in a similar fashion to that of Bortfeldt and Mack using a volume lower bound as shown below for the other datasets (where B is the set of boxes to be packed and cw and ch are the container width and height respectively)

$$VolumeLowerBound = \left\lceil \left(\sum_{b \in B} volume(b) \right) \Big/ (cw \cdot ch) \right\rceil.$$

4.2. Notes on comparison with 2BF

The results on the N1–N13 datasets are not compared against the best known results, instead they are compared against the original 2BF implementation in order to get a clearer idea of how competitive the newer placement policies are against the original policies presented in Burke et al. (2004). The times taken to run the 3BF experiments can be seen to be notably longer than the 2BF counterparts, though this is understandable due to the different methods of representing the problem between the 2BF and 3BF implementation. A two-dimensional array-based method was initially trialled for use in the 3BF implementation but was shown to be too slow for containers with large dimensions (due to the memory allocation and the constant memory traversal needed) and also did not provide the flexibility of being able to handle boxes and containers with floating point dimensions.

4.3. Experimentation and results of 3BF with metaheuristic enhancements

Each run of the algorithm was allowed 160 seconds in total (i.e. including both the deterministic and the metaheuristic phases) to

attempt each problem instance, the same running time as used in Bortfeldt and Mack (2007). For comparison on the N1–N13 datasets a time limit of 60 seconds was imposed to compare more fairly with the experimental set up used in the original paper. The initial value of m was set to 15 for all datasets and was allowed to adapt dynamically. The same rotation restrictions described in Section 4 were imposed and the same test machine used. Note the comparison on the N1–N13 datasets is now with the 2BF heuristic with metaheuristic enhancements (in this case simulated annealing) included as described in Burke et al. (2009). The results included from Burke et al. (2009) are the best results over 10 runs. Results reported by 3BF + TS are averaged over 100 runs, with standard deviations being at, or incredibly close to, zero. This is likely due to near optimal solutions being found (local optima) with little to distinguish between improving solutions. Another evaluation method used at least in the interim for generating the final solution, such as the 'area below the skyline' score given in Burke et al. (2009) could possibly be used to help the search in this respect.

4.4. Evaluation of results

As can be seen in Tables 1–4, the results obtained by pure 3BF are very competitive with the best known results from the literature. In 18 out of 35 cases 3BF achieves the highest volume utilisation currently known (shown in a bold typeface) further to 13 out of 13 from the 2004 Burke et al. paper, and the time taken to do this is considerably less than that of the results from the literature, with the exception of N1–N13 (see Section 4.2).

As with the results of the purely deterministic 3BF shown previously, 3BF with metaheuristic enhancements performs extremely competitively with the datasets from the literature, using a similar running time in each case. As can be seen in Tables 5–8 the best known results were achieved (shown in a bold typeface) in 28 of 35 test instances, and a further 13 best results compared to the paper of Burke et al. (2009).

Table 1
Results on the KI datasets.

Data set	n	3BF		Karabulut and Inceoglu (2004)
		Time (seconds)	Utilisation	Utilisation
C1P1	15	<0.1	**100.0**	**100.0**
C1P2	29	0.1	**100.0**	**100.0**
C1P3	50	0.4	**100.0**	**100.0**
C1P4	106	1.6	**98.5**	98.0
C1P5	155	4.8	88.0	**100.0**
C2P1	16	<0.1	**100.0**	**100.0**
C2P2	25	0.1	**100.0**	92.6
C2P3	52	0.4	97.1	**99.0**
C2P4	100	1.8	87.0	**96.2**
C2P5	151	4.0	**98.0**	96.2
C3P1	21	<0.1	**100.0**	**100.0**
C3P2	31	0.1	92.6	**94.3**
C3P3	51	0.4	87.0	**92.6**
C3P4	101	1.8	89.3	**91.3**
C3P5	151	3.9	82.4	**90.9**
Average			94.7	**96.7**

Table 2
Results on the BR datasets.

Data set	n	3BF		Bortfeldt and Mack (2007)	Bortfeldt and Gehring (1999)
		Time (seconds)	Utilisation	Utilisation	Utilisation
BR1	139	0.3	88.7	87.3	**92.3**
BR2	140	0.4	89.0	88.6	**93.5**
BR3	135	0.5	87.8	89.4	**92.3**
BR4	132	0.6	87.8	90.1	**90.8**
BR5	128	0.7	87.7	89.3	**89.9**
BR6	134	0.9	87.6	**89.7**	89.2
BR7	129	1.1	87.4	**89.2**	87.1
BR8	138	1.3	86.8	**87.9**	84.0
BR9	128	1.6	86.5	**87.3**	80.9
BR10	129	2.0	86.3	**87.6**	79.1
Average			87.6	**88.6**	87.9

Table 3
Results on the BR-XL datasets.

Data set	n	3BF		Bortfeldt and Mack (2007)
		Time (seconds)	Utilisation	Utilisation
BR1-XL	1000	8.6	**92.2**	86.9
BR2-XL	1000	10.2	**92.2**	88.3
BR3-XL	1000	12.5	**91.8**	89.8
BR4-XL	1000	14.9	**92.0**	90.2
BR5-XL	1000	16.8	**92.4**	89.9
BR6-XL	1000	19.4	**92.5**	91.5
BR7-XL	1000	22.9	**92.4**	91.0
BR8-XL	1000	26.2	**92.6**	90.8
BR9-XL	1000	30.1	**92.1**	90.9
BR10-XL	1000	35.3	**92.5**	90.4
Average			**92.3**	90.0

Table 4
Results on the N datasets.

Data set	n	3BF		Burke et al. (2004)	
		Time (seconds)	Utilisation	Time (seconds)	Utilisation
N1	10	<0.1	**100.0**	<0.1	88.9
N2	20	<0.1	**94.3**	<0.1	**94.3**
N3	30	<0.1	**98.0**	<0.1	96.2
N4	40	<0.1	**96.4**	<0.1	**96.4**
N5	50	<0.1	**97.1**	<0.1	95.2
N6	60	0.1	**99.0**	<0.1	97.1
N7	70	0.1	**97.1**	<0.1	93.5
N8	80	0.2	**97.6**	<0.1	95.2
N9	100	0.4	**98.7**	<0.1	**98.7**
N10	200	0.6	**99.3**	<0.1	98.7
N11	300	1.0	**99.3**	<0.1	98.7
N12	500	1.2	**99.0**	<0.1	98.0
N13	3152	8.5	**99.6**	1.4	**99.6**
Average			**98.0**		96.2

Table 5
Results on the KI datasets using metaheuristic enhancements.

Data set	n	3BF + TS Utilisation	Karabulut and Inceoglu (2004) Utilisation
C1P1	15	**100.0**	**100.0**
C1P2	29	**100.0**	**100.0**
C1P3	50	**100.0**	**100.0**
C1P4	106	**99.5**	98.0
C1P5	155	**100.0**	**100.0**
C2P1	16	**100.0**	**100.0**
C2P2	25	**100.0**	92.6
C2P3	52	**100.0**	99.0
C2P4	100	**98.0**	96.2
C2P5	151	**99.0**	96.2
C3P1	21	**100.0**	**100.0**
C3P2	31	**100.0**	94.3
C3P3	51	**98.0**	92.6
C3P4	101	**95.2**	91.3
C3P5	151	**95.5**	90.9
Average		**99.0**	96.7

Table 6
Results on the BR datasets using metaheuristic enhancements.

Data set	n	3BF + TS Utilisation	Bortfeldt and Mack (2007) Utilisation	Bortfeldt and Gehring (1999) Utilisation
BR1	139	90.0	87.3	**92.3**
BR2	140	89.6	88.6	**93.5**
BR3	135	89.0	89.4	**92.3**
BR4	132	88.8	90.1	**90.8**
BR5	128	88.5	89.3	**89.9**
BR6	134	88.6	**89.7**	89.2
BR7	129	88.7	**89.2**	87.1
BR8	138	**88.3**	87.9	84.0
BR9	128	**87.9**	87.3	80.9
BR10	129	**87.9**	87.6	79.1
Average		**88.7**	88.6	87.9

Table 7
Results on the BR-XL datasets using metaheuristic enhancements.

Data set	n	3BF + TS Utilisation	Bortfeldt and Mack (2007) Utilisation
BR1-XL	1000	**92.4**	86.9
BR2-XL	1000	**92.4**	88.3
BR3-XL	1000	**91.9**	89.8
BR4-XL	1000	**92.1**	90.2
BR5-XL	1000	**92.5**	89.9
BR6-XL	1000	**92.6**	91.5
BR7-XL	1000	**92.6**	91.0
BR8-XL	1000	**92.8**	90.8
BR9-XL	1000	**92.3**	90.9
BR10-XL	1000	**92.7**	90.4
Average		**92.4**	90.0

Table 8
Results on the N datasets using metaheuristic enhancements.

Data set	n	3BF + TS Utilisation	Burke et al. (2009) Utilisation
N1	10	**100.0**	**100.0**
N2	20	**100.0**	**100.0**
N3	30	**100.0**	98.0
N4	40	**99.8**	97.6
N5	50	**99.1**	97.1
N6	60	**99.3**	98.0
N7	70	**98.0**	96.2
N8	80	**97.6**	**97.6**
N9	100	**98.7**	**98.7**
N10	200	**99.3**	98.7
N11	300	**99.3**	98.0
N12	500	**99.0**	98.0
N13	3152	**99.6**	**99.6**
Average		**98.9**	98.3

5. Conclusion

This paper presents an efficient three-dimensional hybrid placement strategy, inspired by the ideas of Burke et al. (2009), which tackles the three-dimensional strip packing problem in an incredibly effective manner.

It has been shown that 3BF is a suitably powerful and efficient method of packing boxes within a container for the three-dimensional strip packing problem, and also that with some metaheuristic enhancements (coupled with a DBLF packing strategy) to this solution, quality can be improved even further in a very reasonable amount of execution time. In this paper 3BF alone has provided the best known solutions for 18 of 35 datasets, and when coupled with the metaheuristic enhancements it has provided the best known solutions for 28 of the 35 datasets. Both approaches have led to better solutions than the original 2BF papers.

The 3BF packing algorithm has been shown to be extremely effective over a wide range of problem instances from the literature, and could be directly applied to many other two and three-dimensional packing problems. The proposed methodology could also easily be extended to include other constraints such as balancing and weight distribution due to the flexibility of the proposed strategy. Further improvements to solution quality could also be incorporated with novel evaluation functions which would integrate with the current strategy with minimal changes to the structure of the methodology. The time taken to reach suitably high solution quality with this strategy is very reasonable on realistically sized instances and could be implemented in many industrial settings with relative ease.

References

Allen, S.D., 2010. CrateViewer: Visualising {One,Two,Three}-Dimensional Packings. University of Nottingham. <www.cs.nott.ac.uk/sda>.

Baker, B.S., Coffman Jr., E.G., Rivest, Ronald L., 1980. Orthogonal packings in two dimensions. SIAM Journal on Computing 9 (4), 846–855.

Bansal, N., Han, X., Iwama, K., Sviridenko, M., Zhang, G., 2007. Harmonic algorithm for 3-dimensional strip packing problem. In: SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 1197–1206.

Bischoff, E., Ratcliff, M., 1995. Issues in the development of approaches to container loading. Omega 23, 377–390.

Bortfeldt, A., Gehring, H., 1999. Two metaheuristics for strip packing problems. In: Proceedings band der 5th International Conference of the Decision Sciences Institute, Athen, vol. 2, pp. 1153–1156.

Bortfeldt, A., Mack, D., 2007. A heuristic for the three-dimensional strip packing problem. European Journal of Operational Research 183 (3), 1267–1279.

Burke, E.K., Kendall, G., Whitwell, G., 2004. A new placement heuristic for the orthogonal stock cutting problem. Operational Research 52 (4), 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 on Computing 21 (3), 505–516.

Chazelle, B., 1983. The bottom-left bin-packing heuristic: An efficient implementation. IEEE Transactions on Computers 32 (8), 697–707.

Coffman Jr., E.G., Garey, M.R., Johnson, D.S., 1978. An application of bin-packing to multiprocessor scheduling. SIAM Journal on Computing 7 (1), 1–17.

Coffman Jr., E.G., Garey, M.R., Johnson, D.S., 1984. Approximation algorithms for bin-packing – an updated survey. Springer, Vienna, pp. 49–106.

Crainic, T.G., Perboli, G., Tadei, R., 2008. Extreme point-based heuristics for three-dimensional bin packing. INFORMS Journal of Computers 20 (3), 368–384.

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

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability. W.H. Freeman and Co., San Francisco, CA (A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences).

Glover, F., Laguna, M., 1993. Tabu search. In: Reeves, C.R. (Ed.), Modern Heuristic Techniques for Combinatorial Problems. John Wiley & Sons, Inc., pp. 70–150.

Hopper, E., Turton, B., 2001a. A review of the application of meta-heuristic algorithms to 2d strip packing problems. Artificial Intelligence Review 16 (4), 257–300.

Hopper, E., Turton, B.C.H., 1999. A genetic algorithm for a 2d industrial packing problem. Computers and Industrial Engineering 37 (1–2), 375–378.

Hopper, E., Turton, B.C.H., 2001b. An empirical investigation of meta-heuristic and heuristic algorithms for a 2d packing problem. European Journal of Operational Research 127 (1), 34–57.

Hower, W., Rosendahl, M., Köstner, D., 1996. Evolutionary algorithm design. In: Artificial Intelligence in Design'96. Kluwer Academic Publishers, Dordrecht, Germany, pp. 663–680.

Jansen, K., Solis-Oba, R., 2006. An asymptotic approximation algorithm for 3d-strip packing. In: SODA '06: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm. ACM, New York, NY, USA, pp. 143–152.

Karabulut, K., Inceoglu, M.M., 2004. A hybrid genetic algorithm for packing in 3d with deepest bottom left with fill method. In: ADVIS, pp. 441–450.

Lai, K.K., Chan, J.W.M., 1997. Developing a simulated annealing algorithm for the cutting stock problem. Computers and Industrial Engineering 32 (1), 115–127.

Li, K., Cheng, K.-H., 1990. On three-dimensional packing. SIAM Journal on Computing 19 (5), 847–867.

Li, K., Cheng, K.-H., 1992. Heuristic algorithms for on-line packing in three dimensions. Journal of Algorithms 13 (4), 589–605.

Mack, D., Bortfeldt, A., Gehring, H., 2004. A parallel hybrid local search algorithm for the container loading problem. International Transactions in Operational Research 11 (5), 511–533.

Miyazawa, F.K., Wakabayashi, Y., 2004. Packing problems with orthogonal rotations. In: LATIN, pp. 359–368.
Riff, M., Bonnaire, X., Neveu, B., 2009. A revision of recent approaches for two-dimensional strip-packing problems. Engineering Applications of Artificial Intelligence 22 (4–5), 823–827.
Roberts, S.A., 1984. Application of heuristic techniques to the cutting-stock problem for worktops. Journal of Operational Research Society 35 (5), 369–377.
Walsh, P., 2002. Advanced 3-D Game Programming Using DirectX 8.0. Wordware Publishing Inc., TX, USA.
Wäscher, G., Haußner, H., Schumann, H., 2007. An improved typology of cutting and packing problems. European Journal of Operational Research 127 (3), 1109–1130.