Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 156 (2004) 601–627
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Discrete Optimization

A population heuristic for constrained two-dimensional non-guillotine cutting

J.E. Beasley *

The Management School, Imperial College, 53 Prince's Gate, Exhibition Road, London SW7 2AZ, UK

Received 29 June 2000; accepted 18 December 2002

Abstract

In this paper we present a heuristic algorithm for the constrained two-dimensional non-guillotine cutting problem. This is the problem of cutting a number of rectangular pieces from a single large rectangle so as to maximise the value of the pieces cut. In addition the number of pieces of each type that are cut must lie within prescribed limits. Our heuristic algorithm is a population heuristic, where a population of solutions to the problem are progressively evolved. This heuristic is based on a new, non-linear, formulation of the problem. Computational results are presented for a number of standard test problems taken from the literature and for a number of large randomly generated problems.

Keywords: Constrained two-dimensional non-guillotine cutting; Population heuristic

1. Introduction

The constrained two-dimensional non-guillotine cutting problem is the problem of cutting from a single large planar stock rectangle (L_0, W_0), of length L_0 and width W_0, smaller pieces (L_i, W_i) each of length L_i and width W_i $(i = 1, \ldots, m)$. Each piece i is of fixed orientation (i.e. cannot be rotated); must be cut (by infinitely thin cuts) with its edges parallel to the edges of the stock rectangle (i.e. orthogonal cuts); and the number of pieces of each type i that are cut must lie between P_i and Q_i $(0 \leqslant P_i \leqslant Q_i)$. Each piece i has an associated value v_i and the objective is to maximise the total value of the pieces cut. Without significant loss of generality it is usual to assume that all dimensions (L_i, W_i) $i = 0, 1, \ldots, m$ are integers. To ease the notation in this paper we shall use $M = \sum_{i=1}^{m} Q_i$.

Fig. 1 shows an example cutting pattern. That pattern is a non-guillotine cutting pattern as the smaller pieces cannot be cut from the stock rectangle using guillotine cuts (cuts that run from one edge of a rectangle to the opposite edge, parallel to the other two edges). Fig. 2, by contrast, illustrates a pattern that can be cut using a succession of guillotine cuts, i.e. it is a guillotine cutting pattern.

* Tel.: +44-207-594-9171; fax: +44-171-823-7685.
E-mail address: j.beasley@ic.ac.uk (J.E. Beasley).
URL: http://mscmga.ms.ic.ac.uk/jeb/jeb.html.

0377-2217/$ - see front matter
doi:10.1016/S0377-2217(03)00139-5

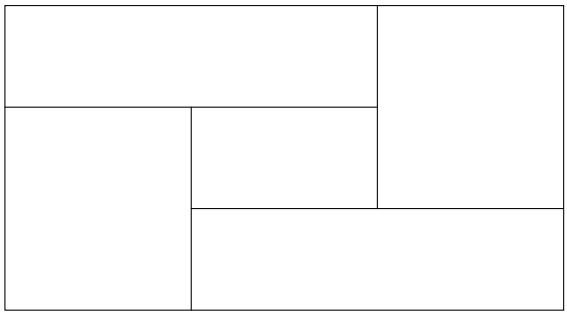

Fig. 1. A non-guillotine cutting pattern.

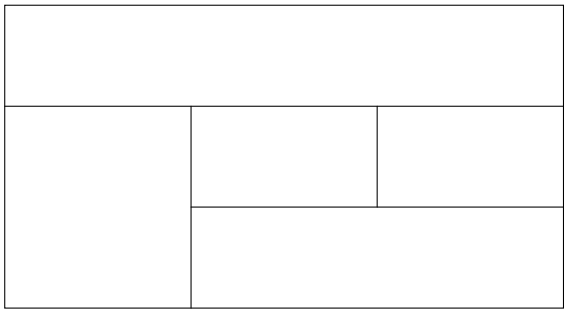

Fig. 2. A guillotine cutting pattern.

A special case of the two-dimensional non-guillotine cutting problem is the pallet loading problem. This occurs when all pieces are of the same size and the name comes from the practical problem of maximising the number of identically sized boxes that can be placed on a wooden pallet. This special case corresponds to $m = 1$ ($v_1 = 1$) when box rotation is not allowed, or to $m = 2$ but with $L_1 = W_2$ and $W_1 = L_2$ ($v_1 = v_2 = 1$) when box rotation is allowed. It is usual in this case to regard the problem as being unconstrained; $P_i = 0$, $Q_i \geqslant (L_0 W_0)/(L_i W_i)$ rounded down to the nearest integer, $i = 1, \ldots, m$.

Two-dimensional cutting problems, of various types, have been widely considered in the literature. Such problems are also commonly referred to as packing problems since they can be viewed:

(a) *either* as the problem of cutting smaller pieces from a larger stock rectangle;
(b) *or* as the problem of packing smaller pieces into a larger stock rectangle.

Dyckhoff [21] has provided a classification of the various types of cutting problem encountered. Dowsland and Dowsland [20], Haessler and Sweeney [25] and Sweeney and Paternoster [36] give a survey of work that has been done prior to the early 1990s.

Relatively few authors in the literature have considered general two-dimensional non-guillotine cutting problems (constrained or unconstrained). With respect to the unconstrained problem work has been presented by Tsai et al. [37]; Arenales and Morabito [3] and Healy et al. [26].

In 1988 Tsai et al. [37] presented an integer programming formulation of the problem. Their approach involves considering the stock rectangle as consisting of a number of horizontal strips each of unit width and length L_0. Each such strip will in a cutting pattern involve pieces whose lengths sum to the length (L_0) of the strip (dummy pieces are included to account for waste). Computationally they only considered one

problem with $m = 3$. Their work contains a number of deficiencies, as Dowsland and Dowsland [20] have commented.

In 1995 Arenales and Morabito [3] presented an approach based upon an AND/OR graph together with branch and bound (tree search). They build up a cutting pattern by making successive cuts, either a single guillotine cut or a set of four non-guillotine cuts that make up a cutting pattern of the type shown in Fig. 1. Their tree search procedure includes heuristics that can be used to prune the tree but, depending upon the parameter values adopted, their use may mean that the optimal solution is not found. Computational results were presented for a number of test problems taken from the literature as well as for a number of randomly generated problems. The largest problem that they solved had $m = 5$.

In 1999 Healy et al. [26] presented an efficient algorithm for identifying the empty space that can be used to cut a new piece when the existing cutting pattern does not have the bottom-left property. A cutting pattern has the bottom-left property when each cut piece is as far down, and as far left, as possible in the stock rectangle. They note that when a cutting pattern with n pieces cut has the bottom-left property the position for a new piece to be cut can be found in $O(n)$ time using an algorithm due to Chazelle [13]. Their paper shows that when a cutting pattern with n pieces cut does not have the bottom-left property the position for a new piece to be cut can be found in $O(n \log(n))$ time. No computational results were given.

With respect to the constrained two-dimensional non-guillotine cutting problem work has been presented by Beasley [5]; Scheithauer and Terno [35]; Hadjiconstantinou and Christofides [24]; Lai and Chan [29,30]; Fekete and Schepers [22,23]; Amaral and Letchford [1]; Leung et al. [31]; and Wu et al. [39].

In 1985 Beasley [5] presented a tree search approach based upon Lagrangean relaxation of a zero–one integer linear programming formulation of the problem to derive an upper bound on the optimal solution. His formulation of the problem uses variables that relate to whether a piece of a particular type (size) is cut with its bottom-left hand corner at a certain position or not. Subgradient optimisation was used in an attempt to minimise the upper bound obtained from the Lagrangean relaxation. Tests to reduce the size of the problem, based both upon the original problem and using upper bound information, were given. He presented computational results for the optimal solution of twelve different test problems with the largest problem solved having $m = 10$.

In 1993 Scheithauer and Terno [35] presented an integer programming formulation of the problem with zero–one variables representing whether a piece is cut to the right/left or above/below another piece. In their formulation of the problem all pieces are cut from an enlarged stock rectangle, but by using zero–one variables only those pieces cut from the original stock rectangle (L_0, W_0) are included in terms of contributing to the objective function. No computational results were given however.

In 1995 Hadjiconstantinou and Christofides [24] used Lagrangean relaxation to derive a bound on the optimal solution for use in a tree search procedure. Note here that their formulation of the problem is incorrect, although the bound given is a valid bound (e.g. see [2]). Subgradient optimisation was used in an attempt to minimise the upper bound obtained from the Lagrangean relaxation. Tests to reduce the size of the problem, based both upon the original problem and using upper bound information, were given. They presented optimal solutions for seven different test problems, as well as the value of the best feasible solution found for a further five test problems which were artificially terminated by a computational time limit constraint. The largest problem that they solved had $m = 15$.

In 1997 Lai and Chan [29] presented a heuristic algorithm for the case $P_i = 0\ \forall i$ based upon simulated annealing. They used a problem representation which encodes the order in which pieces should be cut, specifically for m pieces by permutations (ordered lists) of $\{1, 2, \ldots, M\}$. For example suppose that $M = 5$ then in this representation the ordered list {2,4,5,1,3} would indicate that piece 2 should be cut first, then piece 4, ..., and lastly piece 3. Note here that in this representation each piece can be cut at most once so that to account for any piece i with $Q_i \geqslant 2$ multiple (Q_i) copies of the piece are created, which explains the use of M in $\{1, 2, \ldots, M\}$ above. A difference process algorithm was used to translate from an ordered list into a cutting pattern, effectively each successive piece in the ordered list being cut (having regard to

previously cut pieces) with its bottom-left corner as near as possible to the bottom-left corner of the stock rectangle. New solutions were generated by swopping two pieces in the ordered list and accepting (or rejecting) the swopped solution in a probabilistic manner related to a temperature parameter (as is common in simulated annealing). In their heuristic the temperature parameter also controls whether moves are examined between pieces that are far apart in terms of the order in which they are cut. Computational results were presented for a number of randomly generated test problems and for test problems drawn from a real-life printing example. All of these test problems had the objective of maximising the total area of the cut pieces (minimise trim loss). The size of the largest problem solved is unclear as (unlike most authors) they regard pieces with the same dimensions as distinct (rather than as multiple copies of the same piece). The best that can be said is that their largest problem had $m \leqslant 35$.

Also in 1997 Lai and Chan [30] presented a heuristic algorithm for the case $P_i = 0\ \forall i$ based upon an evolutionary strategy approach. They used the same representation as in Lai and Chan [29]. Their algorithm includes an improvement (hill-climbing) procedure based on dividing the ordered list into active (cut) and inactive (uncut) pieces and seeing if any (currently) uncut pieces can be cut. Their mutation process involves swopping two pieces in the ordered list. Computational results were presented for a number of randomly generated test problems with the objective being to maximise the total area of the cut pieces (minimise trim loss). The largest problem that they solved had $m = 10$.

Further work was presented in 1997 by Fekete and Schepers [22,23] who gave an optimal (tree search) algorithm based upon a graph-theoretic representation of the relative position of pieces in a feasible cutting pattern. In their approach projections of cut pieces are made onto both the horizontal and vertical edges of the stock rectangle. Each such projection is translated into a graph, where the nodes in the graph are the cut pieces and an edge joins two nodes if the projections of the corresponding cut pieces overlap. They show that a cutting pattern is feasible if and only if the projection graphs have certain properties. In their tree search procedure branching to enumerate cutting patterns relates to whether an edge is included in, or excluded from, a projection graph. Upper bounds derived from the solution to knapsack problems, as well as problem reduction tests, are used to curtail the search. Computational results were presented for a number of publicly available test problems as well as for a large number of randomly generated problems. The largest problems that they solved had $m = 33$ [22] and $m = 40$ [23].

In 1999 Amaral and Letchford [1] built upon the work of Tsai et al. [37] and presented a column generation algorithm in order to derive a number of bounds on the optimal solution. They presented computational results for twelve different test problems with the largest problem solved having $m = 10$. One interesting observation in their paper is that they found that the linear programming relaxation of the formulation of the problem given by Scheithauer and Terno [35] was very poor (recall here that, as noted above, no computational results were given in [35]).

In 2001 Leung et al. [31] considered the case $P_i = 0\ \forall i$. They used the ordered representation of Lai and Chan [29] and discussed producing a cutting pattern from it using both the difference process algorithm of Lai and Chan [29] and a standard bottom-left algorithm (e.g. see Jakobs [28]). They showed that there exist problems such that the optimal solution cannot be found by these two approaches. They presented a simulated annealing heuristic with a move corresponding to either swopping two pieces in the ordered representation or moving a single piece to a new position in the representation. They also presented a genetic approach involving five different crossover operators. Computationally they considered eight test problems but detailed results were not presented. The size of the largest problem solved is unclear for the same reasons as for Lai and Chan [29] above and the best that can be said is that their largest problem had $m \leqslant 30$.

In 2002 Wu et al. [39] presented a paper concerned with the problem of deciding whether (or not) it is possible to cut all pieces from a given stock rectangle. This corresponds to a special case, namely $P_i = Q_i\ \forall i$, of the general problem we consider in this paper. They allow pieces to be placed in either orientation. Their algorithm is based upon cutting each new piece from a corner of the current cutting pattern. Since more

than one corner which can be used will often exist they develop a scheme for evaluating which, out of the possible corners, should be used. Computationally they considered a number of randomly generated problems, as well as some test problems taken from the literature. The largest problem that they solved had $m = 97$.

Considering the literature review above it is clear that, over approximately twenty years, only ten papers relating to the problem considered here, constrained two-dimensional non-guillotine cutting, have been published. Obviously given the mass of scientific work that occurs it is impossible for any literature survey to be completely comprehensive but we do believe that the papers considered above accurately reflect the volume of work that has been done. Of these ten papers six are optimal procedures, with the largest problem solved over these six papers having $m = 40$, and four are heuristic procedures, with the largest problem solved over these four papers having $m = 97$. Moreover note here that, as mentioned above, these four heuristic papers only dealt with special cases of the general two-dimensional non-guillotine cutting problem (three papers [29–31] dealt with $P_i = 0\ \forall i$ and one paper [39] dealt with $P_i = Q_i\ \forall i$). By contrast the heuristic presented in this paper deals with the general case.

In this paper we present a new, non-linear, formulation of the constrained two-dimensional non-guillotine cutting problem. Based upon this formulation we present a population heuristic and give computational results for this heuristic on a number of small test problems taken from the literature. In order to gain insight into the performance of our heuristic on larger test problems we give computational results for 630 randomly generated problems of varying types and evaluate the quality of the solution we obtain by reference to an upper bound for the problem. These test problems are made publicly available for use by future workers.

We illustrate that whereas there exist problems for which the heuristics of Lai and Chan [29,30] and Leung et al. [31] can never find the optimal solution the heuristic presented in this paper has no such limitations. Since we consider problems involving $m = 1000$ our paper considers much larger problems (an order of magnitude larger) than have been solved by other workers. In addition we present computational results for general test problems with $P_i > 0$ for some i. In the next section we present our formulation of the problem.

2. Problem formulation

2.1. Formulation

As we have a number (Q_i) of identical copies of each piece i we define:

$$z_{ip} = 1 \quad \text{if the } p\text{th copy } (p = 1, \ldots, Q_i) \text{ of piece } i \text{ is cut from } (L_0, W_0),$$
$$= 0 \quad \text{otherwise,}$$

where the position of any cut piece is taken with reference to the centre of the piece, i.e. let

x_{ip} be the x-coordinate of the centre of the pth copy of piece i;

y_{ip} be the y-coordinate of the centre of the pth copy of piece i.

Fig. 3 illustrates the situation diagrammatically. These centre coordinates are limited by

$$L_i/2 \leqslant x_{ip} \leqslant L_0 - L_i/2, \quad i = 1, \ldots, m;\ p = 1, \ldots, Q_i, \tag{1}$$

$$W_i/2 \leqslant y_{ip} \leqslant W_0 - W_i/2, \quad i = 1, \ldots, m;\ p = 1, \ldots, Q_i. \tag{2}$$

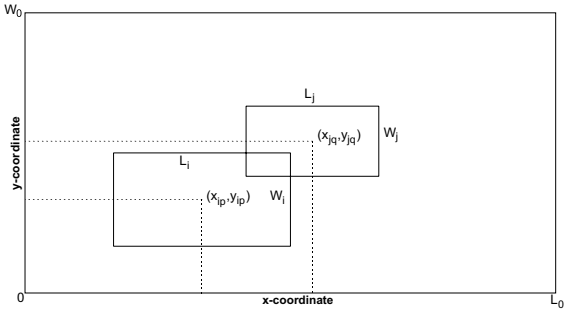

Fig. 3. Overlapping pieces.

In order to ensure that we cut an appropriate minimum number of pieces of each type we have

$$z_{ip} = 1, \quad i = 1, \ldots, m; \; P_i > 0; \; p = 1, \ldots, P_i. \tag{3}$$

This constraint simply says that we (arbitrarily) chose to cut the first P_i copies of piece i.

Naturally pieces that are cut from the stock rectangle (L_0, W_0) cannot overlap and considering Fig. 3 where we have shown two overlapping pieces (the pth copy of i and the qth copy of j) this can be expressed as the condition that we require

$$|x_{ip} - x_{jq}| \geqslant (L_i + L_j)/2 \quad \text{or} \quad |y_{ip} - y_{jq}| \geqslant (W_i + W_j)/2, \tag{4}$$

i.e. that the difference between the centre x-coordinates is sufficient ($\geqslant (L_i + L_j)/2$) to ensure that the pieces do not overlap, *or* that the difference between the centre y-coordinates is sufficient ($\geqslant (W_i + W_j)/2$) to ensure that the pieces do not overlap. This overlap constraint was first given by Christofides [14] in the context of strip packing. To ease the notation define $\alpha_{ij} = (L_i + L_j)/2$ and $\beta_{ij} = (W_i + W_j)/2$, then the above condition (Eq. (4)) can be written as

$$|x_{ip} - x_{jq}| - \alpha_{ij} \geqslant 0 \quad \text{or} \quad |y_{ip} - y_{jq}| - \beta_{ij} \geqslant 0. \tag{5}$$

Hence our constraint to prevent overlap between cut pieces is

$$\max[|x_{ip} - x_{jq}| - \alpha_{ij}, |y_{ip} - y_{jq}| - \beta_{ij}]z_{ip}z_{jq} \geqslant 0, \quad i = 1, \ldots m; \; j = 1, \ldots, m; \; p = 1, \ldots, Q_i; \; q = 1, \ldots, Q_j (i \neq j \text{ or } p \neq q) \tag{6}$$

where the $z_{ip}z_{jq}$ term renders the constraint inactive unless $z_{ip} = z_{jq} = 1$ (i.e. unless both the pth copy of piece i and the qth copy of piece j are cut).

Our complete formulation of the constrained two-dimensional non-guillotine cutting problem is therefore:

$$\text{maximise} \quad \sum_{i=1}^{m} \sum_{p=1}^{Q_i} v_i z_{ip} \tag{7}$$

subject to

$$\max[|x_{ip} - x_{jq}| - \alpha_{ij}, |y_{ip} - y_{jq}| - \beta_{ij}]z_{ip}z_{jq} \geqslant 0, \quad i = 1, \ldots m; \; j = 1, \ldots, m; \; p = 1, \ldots, Q_i; \; q = 1, \ldots, Q_j (i \neq j \text{ or } p \neq q), \tag{8}$$

$$z_{ip} = 1, \quad i = 1, \ldots, m; \; P_i > 0; \; p = 1, \ldots, P_i, \tag{9}$$

$$L_i/2 \leqslant x_{ip} \leqslant L_0 - L_i/2, \quad i = 1, \ldots, m; \; p = 1, \ldots, Q_i, \tag{10}$$

$$W_i/2 \leqslant y_{ip} \leqslant W_0 - W_i/2, \quad i = 1, \ldots, m; \; p = 1, \ldots, Q_i, \tag{11}$$

$$z_{ip} \in (0, 1), \quad i = 1, \ldots, m; \; p = 1, \ldots, Q_i. \tag{12}$$

There are several points to note about this formulation:

(a) If the pth copy of piece i is not cut ($z_{ip} = 0$) then the values of x_{ip} and y_{ip} are irrelevant as the overlap constraint (Eq. (8)) is automatically satisfied.
(b) The overlap constraint as presented above (Eq. (8)) contains redundancy since if we were to write out this overlap constraint in full for any particular example we would find exactly the same mathematical equation appearing twice (e.g for $i = 3$, $p = 1$ and $j = 7$, $q = 2$ we get the same equation as for $i = 7$, $p = 2$ and $j = 3$, $q = 1$). However, we prefer this overlap constraint in the form presented above for later use in our heuristic algorithm.
(c) The formulation given above is valid irrespective of whether the dimensions of the pieces (L_i, W_i), $i = 0, 1, \ldots, m$, are integers or not. This contrasts with the formulation of the problem presented previously in [5] which was dependent on the pieces having integer dimensions.
(d) The centre coordinates (x_{ip}, y_{ip}) can take fractional values; or if we wish to work with integer centre coordinates then without significant loss of generality simply ensure that all dimensions (L_i, W_i), $i = 0, 1, \ldots, m$, are integer and even (set $L_i = 2L_i$ and $W_i = 2W_i$, $i = 0, 1, \ldots, m$).

Our formulation of the problem is a non-linear one, involving as it does in Eq. (8) a maximisation of two modulus terms and the product of two zero–one variables. Using $M = \sum_{i=1}^{m} Q_i$ then our formulation has $3M$ variables and $O(M^2)$ constraints (excluding bounds on variables). In the formulation of this problem given previously in Beasley [5] the number of variables and constraints was a function of the size of the stock rectangle (L_0, W_0). The formulation given above does not, in terms of the number of variables and constraints, involve the size of the stock rectangle. Overall we can say that the formulation presented here is a "more compact" formulation than that given in [5], principally due to the non-linear representation of the overlap constraint in Eq. (8).

Before presenting the population heuristic based upon our formulation we shall briefly indicate how our formulation can be extended to deal with a number of situations encountered in two-dimensional non-guillotine cutting.

2.2. Extensions

In this section we indicate how our formulation can be extended to deal with defects, multiple stock rectangles and piece rotation. Each of these is discussed separately below.

2.2.1. Defects

Defects in the stock rectangle (L_0, W_0), i.e. areas of the stock rectangle which cannot be part of any cut piece, are easily dealt with in our formulation. Suppose, without significant loss of generality, that all defective areas can be decomposed into rectangular shapes, as in Fig. 4. In that figure we have one defective area (the shaded portion) which has been decomposed into the two (overlapping) rectangular shapes shown. Each rectangular defective can be regarded as a rectangular piece which must be cut at a pre-specified position.

To illustrate how our formulation can be extended, suppose we regard one of the rectangular defectives in Fig. 4 as piece k centred at (a_k, b_k). Set $P_k = Q_k = 1$, so ensuring via Eq. (9) that the piece is cut, then the only changes necessary to our formulation are:

(a) to replace the bounds (Eqs. (10) and (11)) on the centre coordinates of piece k by $x_{k1} = a_k$ and $y_{k1} = b_k$; and
(b) not to impose the overlap constraint (Eq. (8)) for pairs of rectangular defectives (e.g. in Fig. 4 we have two overlapping rectangular defectives).

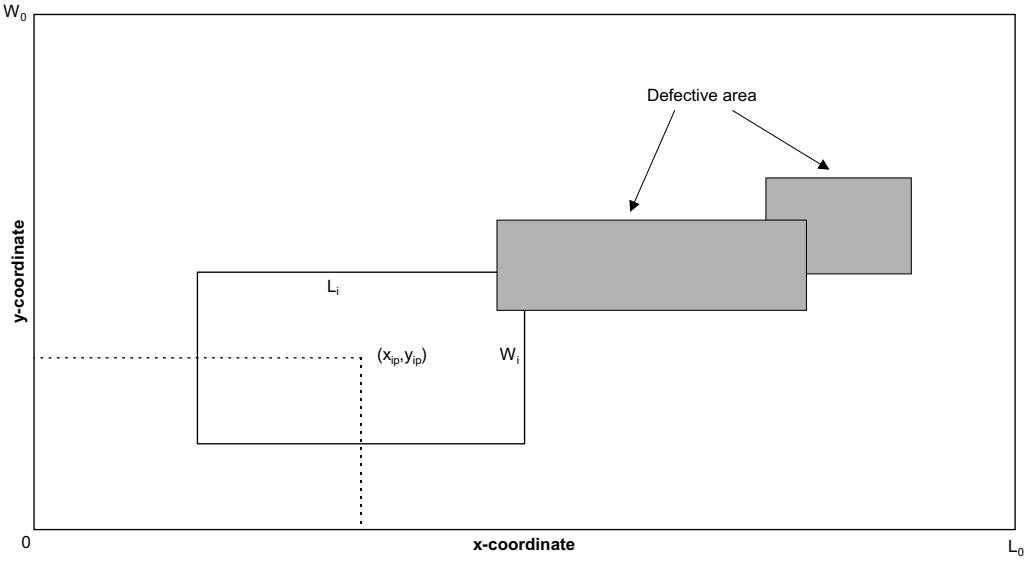

Fig. 4. Defects.

The heuristic we present below is designed such that these changes are easily incorporated.

2.2.2. Multiple stock rectangles

Multiple stock rectangles can be easily dealt with by assembling them, in an arbitrary fashion, into one large rectangular "superstock" separated by defective areas. This is shown diagrammatically in Fig. 5. Note here that, as in Fig. 5, there is no requirement that the individual stock rectangles be of the same size. We can then simply apply our formulation, amended to deal with defective areas as discussed above, to the "superstock" rectangle.

2.2.3. Piece rotation

When pieces have flexible orientation, i.e. can be rotated, then one way to include such flexibility into our formulation is via "paired pieces", each with fixed orientation. For illustration we shall just consider a single piece i which can be rotated. Now regard i as being of fixed orientation with length L_i and width W_i but paired with piece j of length $L_j = W_i$ and width $W_j = L_i$ (i.e. j is a rotation though 90 degrees of i) where

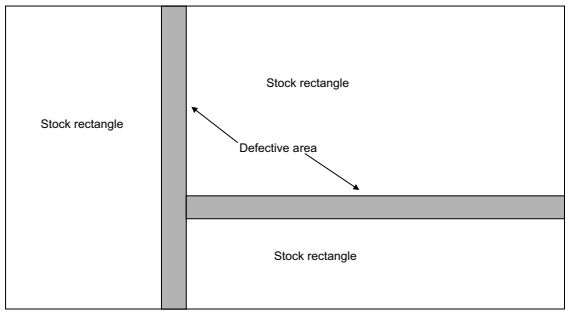

Fig. 5. Multiple stock rectangles.

we have $P_j = P_i$, $Q_j = Q_i$ and $v_j = v_i$. Informally each of the Q_i copies of piece i are paired with the corresponding copy of piece j. Then we need to remove from our formulation Eq. (9) as it applies to pieces i and j, replacing those constraints by

$$z_{ip} + z_{jp} = 1, \quad P_i > 0;\ p = 1, \ldots, P_i, \tag{13}$$

and add to the formulation the constraint

$$z_{ip} + z_{jp} \leqslant 1, \quad p > P_i;\ p = 1, \ldots, Q_i. \tag{14}$$

Eq. (13) ensures that P_i copies of the piece (of either orientation) are cut whilst Eq. (14) ensures that we cannot cut both the piece and its rotated paired copy.

2.3. Summary

In this section we have given our formulation of the constrained two-dimensional non-guillotine cutting problem and have indicated how it can be extended to deal with defects, multiple stock rectangles and piece rotation. In the next section we briefly introduce population heuristics.

3. Population heuristics

In operations research (OR), over the years, a number of standard heuristic algorithms have appeared. In the early years of OR heuristics were especially tailored to the problem under consideration. As OR advanced a number of basic heuristic ideas began to be formalised. These included ideas such as greedy (seek the best possible improvement) and local interchange (e.g. swap two parts of a solution). Further advances came with schemes such as tabu search (accept a change in the solution, for better or worse, unless it is tabu) and simulated annealing (accept a change in the solution, for better or worse, according to some probabilistic criterion). These latter schemes are sometimes called metaheuristics as they provide general guidelines as to the approach to be followed, yet need tailoring for each specific problem under consideration.

All of the above have a common characteristic, namely that they have a single solution to a problem and continually seek to improve this single solution in some way. *Population heuristics*, by contrast, explicitly work with a population of solutions and combine them together in some way to generate new solutions.

Population heuristics started with the development of genetic algorithms by Holland [27]. As the field advanced (evolved) new terms began to appear in the literature: hybrid genetic algorithms, memetic algorithms, scatter search algorithms, bionomic algorithms. Whilst there are differences between these approaches they can all be classified generically as population heuristics.

A population heuristic (henceforth abbreviated to PH) can be described as an "intelligent" probabilistic search algorithm and is based on the evolutionary process of biological organisms in nature. During the course of evolution, natural populations evolve according to the principles of natural selection and "survival of the fittest". Individuals who are more successful in adapting to their environment will have a better chance of surviving and reproducing, whilst individuals who are less fit will be eliminated. This means that the genes from highly fit individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from highly adapted parents may produce even more fit offspring. In this way species evolve to become increasingly better adapted to their environment.

A PH simulates these processes in a computer by taking an initial population of individuals and applying genetic operators in each reproduction. In optimisation terms each individual in the population is encoded into a string or *chromosome* which represents a possible *solution* to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit individuals or *solutions* are

given opportunities to reproduce by exchanging pieces of their genetic information, in a *crossover* procedure, with other highly fit individuals. This produces new "offspring" solutions (i.e. *children*), who share some characteristics taken from both parents. Mutation is often applied after crossover by altering some genes in the child strings. The offspring can either replace the whole population (*generational* approach) or replace less fit individuals (*steady-state* approach). This evaluation-selection-reproduction cycle is repeated until a satisfactory solution is found. The basic steps of a simple PH are:

Generate an initial population.
Evaluate fitness of individuals in the population.
repeat
Select individuals from the population to be parents
Recombine (mate) parents to produce children
Mutate the children
Evaluate fitness of the children
Replace some or all of the population by the children.
until
You decide to stop whereupon report the best solution encountered

Readers wishing to learn more about population heuristics are referred to [4,8,32–34]. For those readers who have some familiarity with PHs we would comment here that one of the difficulties of working with PHs is that there are MANY schemes for doing the basic operations that you need:

- parent selection: binary tournament, K tournament, ranking, fitness proportionate, etc;
- having children: uniform, one-point, restricted one-point, fusion, two-point, etc;
- mutation: constant, adaptive;
- population replacement: steady-state, generational (with/without elitism), island models.

If, in terms of computational experimentation, one were to try all possible combinations even the restricted list of options given above yields $4 \times 5 \times 2 \times 4 = 160$ different possibilities (4 choices for parent selection, 5 choices for having a child, etc). This is before one explores values for necessary numeric parameters such as population size and mutation probability.

Our experience has been that, unless one is careful, it is easy to "drown in possibilities" when trying to develop a PH. In order to avoid exhaustive examination of a large number of combinations of possible operations we have limited ourselves to those operations outlined in the following sections. Those choices have been made in the light of our experience [9–12,16–18] in developing PHs and limited computational experimentation with different choices for various operations.

4. A population heuristic for two-dimensional cutting

In this section we will outline the population heuristic that we have developed for the constrained two-dimensional non-guillotine cutting problem, Below we discuss the elements of our PH relating to:

- representation, how we represent a solution to the problem in our PH;
- fitness, assessing the value of a solution;
- parent selection, choosing who shall have a child;
- crossover, having a child from parents;
- mutation, altering the child;

- unfitness, dealing with the constraints;
- improvement, improving the child; and
- population replacement, placing the child in the population.

4.1. Representation and fitness

The first decision in any PH is how to represent a solution to the problem. For the constrained two-dimensional non-guillotine cutting problem we used a combined binary/real-numbered representation.

The binary part of our representation is simply z_{ip}, which takes the value zero or one and indicates whether the pth copy of piece i is cut or not.

Letting X_{ip} ($0 \leqslant X_{ip} \leqslant 1$) be the real-numbered representation of the x-coordinate value for the centre of the pth copy of piece i then X_{ip} represents the proportion of the interval $[L_i/2, L_0 - L_i/2]$ that lies before x_{ip}, i.e. $x_{ip} = L_i/2 + X_{ip}(L_0 - L_i)$. For example if $X_{ip} = 0.73$, $L_0 = 10$ and $L_i = 4$ then the x-coordinate of the centre of this piece lies 0.73 of the way between 2 ($L_i/2$) and 8 ($L_0 - L_i/2$), i.e. $x_{ip} = 2 + 0.73(10 - 4) = 6.38$ (rounded to the nearest integer 6 as we chose to work with integer centre coordinates).

In a similar fashion we let Y_{ip} ($0 \leqslant Y_{ip} \leqslant 1$) be the real-numbered representation of the y-coordinate value for the centre of the pth copy of piece i so that Y_{ip} represents the proportion of the interval $[W_i/2, W_0 - W_i/2]$ that lies before y_{ip}, i.e. $y_{ip} = W_i/2 + Y_{ip}(W_0 - W_i)$. For example if $Y_{ip} = 0.21$, $W_0 = 30$ and $W_i = 6$ then the y-coordinate of the centre of this piece lies 0.21 of the way between 3 ($W_i/2$) and 27 ($W_0 - W_i/2$), i.e. $y_{ip} = 3 + 0.21(30 - 6) = 8.04$ (rounded to the nearest integer 8).

In order to clarify our representation we show below a sample complete representation for a problem with $m = 2$, $Q_1 = 3$ and $Q_2 = 2$:

Piece (i)	1	1	1	2	2
Copy (p)	1	2	3	1	2
z_{ip}	1	0	0	1	0
X_{ip}	0.73	0.34	0.27	0.56	0.49
Y_{ip}	0.21	0.97	0.88	0.94	0.11

In this example the first copy of piece 1 is cut ($z_{11} = 1$) with its centre coordinates at the appropriate integer position revealed by decoding $X_{11} = 0.73$ and $Y_{11} = 0.21$. The only other piece cut is the first copy of piece 2, at the appropriate integer position revealed by decoding X_{21} and Y_{21}.

The representation we have adopted has the property that we automatically ensure that the constraints on the centre coordinate positions (Eqs. (10) and (11)) are satisfied. Note here that if defective rectangles (e.g. piece k, copy 1) are present, as discussed above, we simply decode any values of X_{k1} and Y_{k1} to be the pre-specified centre coordinates of the defective rectangle.

Fitness in PHs relates to assessing the value (worth) of an individual. In our PH for the constrained two-dimensional non-guillotine cutting problem the fitness of an individual was given by the objective function value (Eq. (7)), which we are trying to maximise.

4.2. Parent selection, crossover and mutation

In our PH we have two parents coming together to have a single child. Our two parents were chosen randomly from the population. In order to produce a child from these two parents we used uniform crossover. This works by considering each element in our representation in turn (i.e. each copy of each

piece) and choosing values taken from either the first or second parent at random. Taking our example from before if we have:

	Piece (i)	1	1	1	2	2
	Copy (p)	1	2	3	1	2
Parent 1	z_{ip}	1	0	0	1	0
	X_{ip}	0.73	0.34	0.27	0.56	0.49
	Y_{ip}	0.21	0.97	0.88	0.94	0.11
Parent 2	z_{ip}	0	1	0	1	1
	X_{ip}	0.13	0.45	0.32	0.99	0.34
	Y_{ip}	0.23	0.43	0.85	0.57	0.69

and our random choice of parents is:

Piece 1 copy 1—parent 1
Piece 1 copy 2—parent 2
Piece 1 copy 3—parent 2
Piece 2 copy 1—parent 1
Piece 2 copy 2—parent 2

then the child is:

Piece (i)	1	1	1	2	2
Copy (p)	1	2	3	1	2
z_{ip}	1	1	0	1	1
X_{ip}	0.73	0.45	0.32	0.56	0.34
Y_{ip}	0.21	0.43	0.85	0.94	0.69

Mutation in our PH was applied to only one in ten children. If a child was selected for mutation then a randomly selected piece/copy (piece j, copy q, say) in that child that had $z_{jq} = 1$ (i.e. was being cut) had its z_{jq} value set to zero (i.e. it was no longer cut). If the area of the cut pieces in the mutated child exceeded L_0W_0 then this process was repeated until the area of the cut pieces in the mutated child was less than (or equal to) L_0W_0 . The logic here was that mutation was directed towards having cut pieces which via the improvement scheme (see below) could be adjusted into a feasible cutting pattern. Plainly if the area of the cut pieces in the child exceeds L_0W_0 then no feasible cutting pattern exists.

4.3. Unfitness

Unfitness in a PH relates to dealing with the constraints. Considering our formulation of the problem given above (Eqs. (7)–(12)) we have that the constraints on the centre coordinates (Eqs. (10) and (11)) are automatically satisfied by virtue of the representation we have adopted (as remarked previously). This leaves two sets of constraints, Eq. (8) which is the overlap constraint and Eq. (9) which ensures that we cut an appropriate minimum number of pieces of each type.

To deal with Eq. (9) we apply a simple adjustment heuristic to the child. Simply consider each piece i in turn in the child and set $z_{ip} = 1\ \forall p \leqslant P_i$. This simple heuristic will automatically ensure that Eq. (9) is satisfied by the child.

In order to deal with the overlap constraint (Eq. (8)) we associate with each individual in our PH two numbers: (*unfitness*, *fitness*) where fitness is our adopted objective function value (which we are trying to maximise) and unfitness is a measure of constraint violation.

Considering Eq. (8) we have that a convenient measure of constraint violation is given by

$$\sum_{i=1}^{m} \sum_{\substack{j=1 \\ i \neq j \text{ or } p \neq q}}^{m} \sum_{p=1}^{Q_i} \sum_{q=1}^{Q_j} \max[0, -\max[|x_{ip} - x_{jq}| - \alpha_{ij}, |y_{ip} - y_{jq}| - \beta_{ij}] z_{ip} z_{jq}] \tag{15}$$

which will be zero if the solution is feasible, non-zero (strictly positive) otherwise. This quantity is the *unfitness* associated with any PH solution and is a measure of constraint infeasibility. Informally the higher the unfitness value the more infeasible a solution.

This use of (unfitness, fitness) allows us to design our PH to evolve feasible solutions (i.e. solutions with unfitness zero) through an appropriate population replacement scheme (see below).

Note here that if defective rectangles are present (as discussed above) we simply amend our definition of unfitness to exclude from Eq. (15) any contribution from pairs of defective rectangles (since these may overlap by definition, see Fig. 4).

In a similar fashion if piece rotation is allowed then we add to Eq. (15) terms reflecting any unfitness contribution from violation of the paired copy constraints (Eqs. (13) and (14)) relating to rotated pieces. Considering those equations a convenient measure of constraint violation is given by

$$\sum_{p=1}^{P_i} |(z_{ip} + z_{jp}) - 1| + \sum_{p=P_i+1}^{Q_i} \max[0, (z_{ip} + z_{jp}) - 1]. \tag{16}$$

Once unfitness has been amended to include Eq. (16) then our heuristic can be easily modified and applied.

4.4. Improvement

Given a child we may be able to improve it (reduce its unfitness if it is infeasible). In order to do this we take the centre coordinates of each piece/copy (piece i, copy p, say) that is cut (i.e. has $z_{ip} = 1$, with its centre at (x_{ip}, y_{ip})) and examine moving the piece/copy both left/right and up/down. Pieces/copies are considered in order of their closeness to the bottom-left corner of the stock rectangle, i.e. in increasing $\sqrt{[(x_{ip}/L_0)^2 + (y_{ip}/W_0)^2]}$ order.

There are four possible simple moves for the centre (x_{ip}, y_{ip}) of the pth copy of piece i:

(a) $x_{ip} = x_{ip} - 1$ (move left);
(b) $y_{ip} = y_{ip} - 1$ (move down);
(c) $x_{ip} = x_{ip} + 1$ (move right);
(d) $y_{ip} = y_{ip} + 1$ (move up).

These moves are considered in the order given above. With respect to the first two of these moves if they reduce (or leave unchanged) the unfitness then the move is made. The last two of these moves are only made if they reduce the unfitness. The reason for this distinction here is to prevent cycling and to seek a "bottom-left" cutting pattern. These moves are repeated for each cut piece/copy in turn until no further improvement in unfitness can be made.

Note here that some moves may not be possible (e.g. if $x_{ip} = L_i/2$ then it is not possible to move the piece/copy left). Note too here that a naive evaluation of unfitness (Eq. (15)) for the child after a move would require $O(M^2)$ operations. However, by recording the contribution to unfitness made by each piece/copy, it is possible to re-evaluate unfitness after a move in just $O(M)$ operations.

After the above improvement procedure the child may be feasible (have unfitness zero, and be a bottom-left cutting pattern). Such a child may improve upon the best feasible solution found previously. In the event that this happens then we seek an even better feasible solution by cutting, from any "empty spaces", pieces that are currently uncut. However, in order not to perturb the convergence of our PH, we do not alter the child to include pieces which are cut from such empty spaces.

4.5. Population replacement

We used a steady-state population replacement scheme that makes use of the two numbers (unfitness,fitness) associated with each PH solution. Suppose that we have produced a child in our PH. This child will have certain (unfitness,fitness) values. For the purposes of illustration we shall assume that this child has unfitness > 0, i.e. is not feasible.

In order to explain our population replacement scheme suppose we draw a graph with fitness as the vertical axis and unfitness as the horizontal axis, as in Fig. 6. Now plot on this graph:

(a) the PH child; and
(b) each member of the population, as each population member will also have (unfitness,fitness) values. For the purposes of illustration these population members are shown as square boxes in Fig. 6, scattered around the child.

Considering Fig. 6 it is clear that the presence of the child has naturally divided the population into four groups corresponding to each of the four quadrants created by drawing horizontal and vertical lines through the child. These four groups are labelled G1–G4 in Fig. 6. Informally it is clear that:

(a) our child is superior to any member of the population that is in group G1, as all members of that group are more infeasible (have a higher unfitness) and are worse in objective function terms (have a lower fitness);
(b) our child is inferior to any member of the population that is in group G4, as all members of that group are less infeasible (have a lower unfitness) and are better in objective function terms (have a higher fitness);
(c) our child is superior with respect to one measure (unfitness or fitness), but inferior with respect to the other, when compared to members of groups G2 or G3.

Now as we have a child we need to add this child to the population. To keep the population size constant (the standard PH assumption) we need to choose a member from the population to kill (i.e. to replace the chosen member by the child).

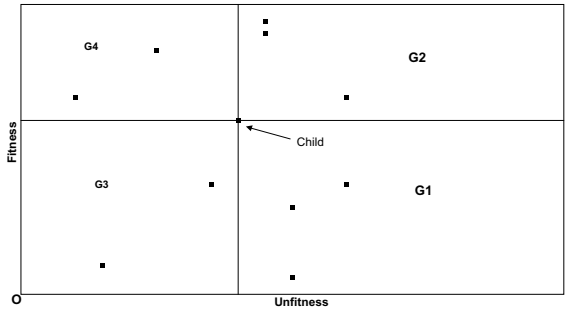

Fig. 6. Population replacement.

We use our four groups (G1–G4) in order to establish in which "area" of the population we look to find a member to kill. In our population replacement scheme we:

(a) first look in group G1. As stated above any population member in this group is worse than the child on both measures (fitness and unfitness). Hence we should plainly choose a member of the population from this group to kill if we can do so;
(b) it may be however that G1 is empty (i.e. contains no population members). If this is the case we look in G2, members of this group are inferior to the child in terms of unfitness;
(c) if both G1 and G2 are empty we look in G3, members in this group are inferior to the child in terms of fitness;
(d) finally if G1–G3 are all empty we look in G4. Members of this group are, as mentioned above, superior to the child on both measures (unfitness and fitness). It might appear strange therefore to choose a member of this group to kill. However, computational experience with this population replacement scheme has been that unless we perturb the population by placing the child in the population the PH fails to make progress.

Once the appropriate group (G1, G2, G3 or G4) has been identified as outlined above we kill a randomly selected member of the group.

There is one complication that must be mentioned here, namely that a child could be a duplicate of a member of the population. By duplicate we mean have exactly the same numeric representation, or have a representation that in terms of the underlying cutting pattern is equivalent. For example consider:

	Piece (i)	1	1	1	2	2
	Copy (p)	1	2	3	1	2
Child	z_{ip}	0	1	0	1	0
	X_{ip}	0.23	0.73	0.23	0.56	0.49
	Y_{ip}	0.18	0.21	0.88	0.94	0.11
Population	z_{ip}	1	0	0	0	1
member	X_{ip}	0.73	0.34	0.55	0.23	0.56
	Y_{ip}	0.21	0.97	0.83	0.48	0.94

Here the child is a duplicate of this population member since the cut pieces associated with each (and their positions) are identical. In order to prevent the population coming to consist of copies of the same cutting pattern any child which was a duplicate of any member of the population was rejected and was not allowed to enter the population.

To assist in duplicate identification, as well as in crossover, the copies of each piece i in the representation were sorted by decreasing z_{ip} values, ties broken firstly by increasing X_{ip} values and secondly by increasing Y_{ip} values. For the example given above the child and population member after such sorting would be:

	Piece (i)	1	1	1	2	2
	Copy (p)	1	2	3	1	2
Child	z_{ip}	1	0	0	1	0
	X_{ip}	0.73	0.23	0.23	0.56	0.49
	Y_{ip}	0.21	0.18	0.88	0.94	0.11
Population	z_{ip}	1	0	0	1	0
member	X_{ip}	0.73	0.34	0.55	0.56	0.23
	Y_{ip}	0.21	0.97	0.83	0.94	0.48

It can be seen here how this sorting helps to more clearly identify duplicates.

4.6. Summary

Our complete PH for the constrained two-dimensional non-guillotine cutting problem can be summarised as follows:

(a) Generate an initial population. In the computational results given below we used a randomly generated initial population with a population size of 100.
(b) Evaluate fitness and unfitness of individuals in the population.
repeat
(c) Select two individuals from the population to be parents, parents were randomly selected.
(d) Recombine (mate) parents to produce a single child, parents were combined using uniform crossover.
(e) Mutate the child, one in ten children were mutated.
(f) Evaluate the fitness and unfitness of the child and improve the child using the heuristic improvement scheme.
(g) Replace one member of the population by the child using the population replacement scheme discussed, unless the child is a duplicate of a member of the population in which case reject it.
until
(h) You decide to stop whereupon report the best solution encountered.

In the computational results reported below we repeatedly applied (replicated) this PH in the following manner:

(1) from an initial population apply the procedure given above until 5000 children have been generated; and then
(2) restart the PH from scratch using a new initial population.

5. Computational results

5.1. Publicly available small test problems

Our population heuristic was coded in FORTRAN and run on a Silicon Graphics O2 workstation (R10000 chip, 225 MHz, 128 MB main memory). In order to examine the effectiveness of our PH we solved 21 publicly available small test problems for which optimal solutions are known. These test problems were:

- twelve test problems given in Beasley [5] which are publicly available from OR-Library [6,7], email the message *ngcutinfo* to o.rlibrary@ic.ac.uk or see http://mscmga.ms.ic.ac.uk/jeb/orlib/ngcutinfo.html;
- two test problems given in Hadjiconstantinou and Christofides [24];
- one test problem given in Wang [38];
- one test problem given in Christofides and Whitlock [15];
- five test problems given in Fekete and Schepers [22,23].

Our results are shown in Table 1. As the optimal solutions for these test problems are known we adopted the approach of terminating our PH when the optimal solution was discovered, or after 15 replications (75 000 children in total) of our heuristic. In Table 1 we show, for each test problem, the size of the problem, the optimal solution, the best solution found by our heuristic, the number of children generated before this best solution was found, the total number of children generated and the total computation time (in seconds).

It is clear from Table 1 that for the smaller problems the optimal solution is discovered very quickly. For the larger problems in that table good quality solutions are found in reasonable computation times. Over all

Table 1
Computational results

Problem source	Problem instance	Problem size			Optimal solution	Best solution		Termination	
		(L_0, W_0)	m	M		Value	Number of children	Number of children	Time (seconds)
Beasley [5]	1	(10,10)	5	10	164	Optimal	38	38	0.02
	2		7	17	230	Optimal	1464	1464	0.16
	3		10	21	247	Optimal	5539	5539	0.53
	4	(15,10)	5	7	268	Optimal	1	1	0.01
	5		7	14	358	Optimal	1233	1233	0.11
	6		10	15	289	Optimal	5062	5062	0.43
	7	(20,20)	5	8	430	Optimal	1	1	0.01
	8		7	13	834	Optimal	40069	40069	3.25
	9		10	18	924	Optimal	20034	20034	2.18
	10	(30,30)	5	13	1452	Optimal	275	275	0.03
	11		7	15	1688	Optimal	16066	16066	0.60
	12		10	22	1865	1801	25504	75000	3.48
Hadjiconstantinou and Christofides [24]	3	(30,30)	7	7	1178	Optimal	57	57	0.03
	11		15	15	1270	Optimal	152	152	0.04
Wang [38]		(70,40)	20	42	2726	2721	5408	75000	6.86
Christofides and Whitlock [15]	3	(40,70)	20	62	1860	1720	70591	75000	8.63
Fekete and Schepers [22,23]	1	(100,100)	15	50	27718	27486	1517	75000	19.71
	2		30	30	22502	21976	61057	75000	13.19
	3		30	30	24019	23743	31058	75000	11.46
	4		33	61	32893	31269	31110	75000	32.08
	5		29	97	27923	26332	21755	75000	83.44
Average percentage deviation from optimality, average number of children and average time						1.24%	16095	32857	8.87

test problems the average percentage deviation from optimality, 100(optimal solution value—best solution)/(optimal solution value), is 1.24% in an average computation time of 8.87 seconds. For the eight test problems in Table 1 for which the optimal solution was not discovered the average percentage deviation from optimality is 3.26% in an average computation time of 22.36 seconds.

One aspect of our overall heuristic that is of interest is whether the PH is itself adding value. We could argue that the results from our heuristic are solely due to our child improvement scheme. To investigate this issue we randomly generated child solutions (each child being feasible in terms of the total area of cut pieces being $\leqslant L_0 W_0$) and subjected each child to our improvement scheme. We did this for all of our 21 test problems and set a time limit for each problem equal to 10 multiplied by the time given in Table 1 for our PH. The results were that for only six problems (problems 4, 5, 7 and 10 from Beasley [5]; problems 3 and 11 from Hadjiconstantinou and Christofides [24]) was the optimal solution discovered. For the other 15 test problems the solutions obtained were worse than those given by our PH in Table 1.

5.2. Different choices

As commented above one of the difficulties of working with PHs is the numerous choices that are possible with regard both to the basic operations that are necessary (e.g with regard to parent selection and

the population replacement scheme) and with regard to numeric parameters (e.g. population size). In this section we will investigate this issue.

In order to structure our investigation we will consider just the eight problems shown in Table 1 for which the PH presented above did not find the optimal solution and make a single change to our PH (i.e. make a different choice for one single particular element of our PH). The five different choices, denoted C1–C5, we investigate relate to:

- *Parent selection:* instead of parent selection being random (as in our PH) consider parent selection via binary tournament selection based upon maximum fitness (choice C1) and via binary tournament selection based upon minimum unfitness (choice C2).
- *Population replacement:* of the group chosen (as in our PH above) instead of selecting a random member of the group (as in our PH) select the member with the maximum unfitness, ties broken by minimum fitness (choice C3).
- *Population size:* instead of a population size of 100 (as in our PH) consider a population size of 50 (choice C4) and a population size of 200 (choice C5).

The results are shown in Table 2. In that table we give the results for our PH (taken from Table 1) with regard to the best solution found and the total time taken. We also give the corresponding results for the various different choices outlined above.

It is clear from Table 2 that, excluding choice C1, different choices require broadly the same computation time and give percentage deviations from optimality ranging between 3.26% (for our PH as presented above) and 4.45% (for choice C4). Given the stochastic nature of our heuristic this indicates that our results are reasonably robust with respect to the different choices that can be made.

5.3. Larger problems—problem generation

In order to investigate the performance of our heuristic on larger problems we randomly generated 630 test problems in the same manner as Fekete and Schepers [23].

These test problems involved a stock rectangle (L_0, W_0) of size (100,100). For each value of m considered (where we considered $m = 40$; 50; 100; 150; 250; 500; 1000) ten different test problems were randomly generated with $P_i = 0$, $Q_i = Q^* \ \forall i$ (where we considered $Q^* = 1$; 3; 4).

With respect to piece sizes Fekete and Schepers [23] considered four classes of size, as below:

Class	Description	Length distribution	Width distribution
1	Bulky in width	[1,50]	[75,100]
2	Bulky in length	[75,100]	[1,50]
3	Large	[50,100]	[50,100]
4	Small	[1,50]	[1,50]

For example a piece of class 2 has its length generated by sampling from the uniform distribution [75,100] and its width generated by sampling from the uniform distribution [1,50]. Fekete and Schepers [23] explicitly state that these sample values are rounded up to integer values. A moment's thought will reveal that rounding up actually means that it is extremely unlikely that any piece will be generated with length (or width) equal to the lower limit in the length (or width) distribution. However, for the sake of consistency, we have produced our test problems in accordance with the description they give. The value assigned to a piece is set equal to its area multiplied by an integer random number chosen from {1, 2, 3}.

Fekete and Schepers [23] considered three types of problem (types I–III) where the difference between types relates to the percentage of pieces of each class. Specifically:

Table 2
Computational results—different choices

Problem source	Problem instance	Problem size			Optimal	PH		C1		C2		C3		C4		C5	
		(L_0, W_0)	m	M		Best	Time (seconds)	Best	Time (seconds)	Best	Time (seconds)	Best	Time (seconds)	Best	Time (seconds)	Best	Time (seconds)
Beasley [5]	12	(30,30)	10	22	1865	1801	3.48	1746	3.60	1801	3.57	1801	3.69	1801	2.52	1801	5.25
Wang [38]		(70,40)	20	42	2726	2721	6.86	2721	6.54	2702	6.33	2623	6.89	2702	5.63	2721	8.89
Christofides and Whitlock [15]	3	(40,70)	20	62	1860	1720	8.63	1740	9.25	1680	8.39	Optimal	0.79	1680	7.43	1820	11.44
Fekete and Schepers [22,23]	1	(100,100)	15	50	27718	27486	19.71	26090	7.24	27261	20.75	27486	31.79	26964	14.01	27486	24.28
	2		30	30	22502	21976	13.19	n/f	4.06	21122	12.66	21248	15.30	20642	9.39	20805	16.81
	3		30	30	24019	23743	11.46	n/f	3.79	23743	11.50	23740	13.37	23743	8.24	23454	14.37
	4		33	61	32893	31269	32.08	31034	10.67	30678	27.32	30943	35.66	31640	23.84	31269	38.17
	5		29	97	27923	26332	83.44	23867	13.21	26332	70.17	26332	81.74	26332	69.30	26332	82.47
Average percentage deviation from optimality and average time						3.26%	22.36	–	7.30	4.42%	20.09	3.30%	23.65	4.45%	17.55	3.39%	25.21

Note: n/f means that no feasible solution was found.

Type	Percentage of pieces of class			
	1	2	3	4
I	20	20	20	40
II	15	15	15	55
III	10	10	10	70

For example a problem of type I has 20% of pieces of class 1, 20% of class 2, 20% of class 3 and the remainder (40%) of class 4.

The key parameter here is the percentage of class 4 pieces (small pieces). Fekete and Schepers [23] report that for their algorithm solution difficulty grows with the percentage of small pieces (i.e. type II problems are harder to solve than type I problems and in turn type III problems are harder to solve than type II problems). Since pieces of classes 1 to 3 are bulky or large this is not surprising since there are naturally more possibilities for cutting small (class 4) pieces than for bulky/large pieces.

In summary then in our 630 test problems we have 210 problems of each type made up from ten test problems for each of 21 (m, Q^*) pairs where $m = 40$; 50; 100; 150; 250; 500; 1000 and $Q^* = 1$; 3; 4. All of these test problems are made publicly available for use by future workers (see http://mscmga.ms.ic.ac.uk/jeb/orlib/ngcutinfo.html). Problems of the size considered here are significantly larger than those considered previously in the literature.

In order to gain insight into the quality of solution produced by our heuristic we would ideally like to know the optimal solution for each test problem. As will become apparent from our discussion below however it is clear that problems of the size considered here are well beyond the solution range of the best of the optimal solution algorithms [23].

However it is possible to gain insight into the quality of solution produced by our heuristic by making use of an upper bound on the optimal solution obtained by solving a (simple) integer program. Let r_i = the number of pieces of type i cut from the stock rectangle (L_0, W_0) where $r_i \geqslant 0$ and integer. Then consider the integer program

$$\text{maximise} \quad \sum_{i=1}^{m} v_i r_i \tag{17}$$

$$\text{subject to} \quad \sum_{i=1}^{m} (L_i W_i) r_i \leqslant L_0 W_0, \tag{18}$$

$$P_i \leqslant r_i \leqslant Q_i, \quad i = 1, \ldots, m, \tag{19}$$

$$r_i \geqslant 0 \quad \text{and integer}, \quad i = 1, \ldots, m. \tag{20}$$

Eq. (18) ensures that the total area of the pieces cut does not exceed the area of the stock rectangle and Eq. (19) ensures that the number of pieces of each type cut lies within the appropriate limits. Here the objective (Eq. (17)) is to maximise the value of the pieces cut and the solution to this program will be an upper bound on the optimal solution to the original two-dimensional non-guillotine cutting problem (Eqs. (7)–(12)).

Eqs. (17)–(20) can be easily transformed into a standard knapsack problem and solved computationally in pseudo-polynomial time using dynamic programming. This was done for each of our 630 test problems so that for each test problem we could calculate

$$100(\text{upper bound value} - \text{best PH solution value})/(\text{upper bound value}) \tag{21}$$

which indicates the percentage deviation from the upper bound value (as calculated from (17)–(20)) of the best PH solution found. It is simple to show that this percentage deviation is itself an upper bound on the percentage deviation that we would find were we able to calculate

$$100(\text{optimal solution value} - \text{best PH solution value})/(\text{optimal solution value}). \tag{22}$$

For example suppose that the value calculated from Eq. (21) for a particular test problem is 3%. Then the fact that this is an upper bound on the value that would be calculated for this test problem from Eq. (22) means that we have an absolute guarantee that the best PH solution that has been found deviates from the optimal solution for this test problem by no more than 3%.

5.4. Larger problems—results

Table 3 shows the results for our PH on the larger test problems generated as discussed above. In that table we show for each (m, Q^*) pair:

- the average percentage deviation from the upper bound of the best PH solution found (as calculated using Eq. (21));
- the average replication at which the best PH solution was found (15 replications were performed for each test problem);
- the average computation time (in seconds);

where the average is computed over the ten test problems solved for each (m, Q^*) pair.

Table 3
Computational results—larger problems

m	Q^*	M	Type I			Type II			Type III		
			Average % deviation	Average replication	Average time (seconds)	Average % deviation	Average replication	Average time (seconds)	Average % deviation	Average replication	Average time (seconds)
40	1	40	6.385	6.9	8.9	8.680	5.0	12.7	7.792	7.1	19.1
	3	120	4.706	5.8	33.4	3.071	4.7	38.9	2.849	5.7	70.0
	4	160	2.983	6.1	38.5	3.070	4.8	57.4	3.661	7.7	94.0
50	1	50	5.192	3.3	11.8	5.965	6.5	13.3	5.279	7.8	18.7
	3	150	2.539	6.5	36.2	2.120	3.9	49.7	2.389	5.2	91.9
	4	200	2.679	6.8	60.9	2.776	4.6	71.5	2.421	4.8	107.8
100	1	100	1.979	2.5	19.1	2.390	5.0	25.4	2.419	4.9	37.1
	3	300	1.214	5.2	79.3	1.278	3.0	104.7	1.307	7.0	174.1
	4	400	1.134	9.7	114.1	1.251	5.7	162.1	0.801	4.7	249.1
150	1	150	1.062	8.2	31.6	1.248	5.0	41.5	1.605	4.9	48.7
	3	450	0.609	5.5	149.7	0.520	5.7	166.4	0.666	8.4	255.5
	4	600	1.105	6.3	238.3	0.852	7.2	305.0	0.812	7.9	428.2
250	1	250	0.829	5.7	58.6	0.952	7.3	72.4	0.861	4.0	99.1
	3	750	0.694	7.1	336.9	0.437	5.4	418.0	0.588	8.5	563.5
	4	1000	0.440	7.9	515.1	0.291	8.0	661.5	0.427	7.7	904.4
500	1	500	0.218	4.9	169.5	0.280	4.4	207.6	0.289	6.3	232.2
	3	1500	0.228	10.1	964.6	0.115	6.8	1233.2	0.204	6.9	1434.6
	4	2000	0.180	7.1	1452.7	0.144	5.3	1728.0	0.208	8.7	2191.8
1000	1	1000	0.094	7.2	607.6	0.088	7.5	658.6	0.086	7.8	735.5
	3	3000	0.061	8.3	2832.6	0.040	5.0	3248.2	0.098	7.1	3874.6
	4	4000	0.039	6.8	3961.0	0.075	6.7	4760.5	0.089	6.4	5800.2
Average			1.637	6.6	558.1	1.697	5.6	668.4	1.660	6.6	830.0

Considering Table 3 it is clear that average percentage deviation from the upper bound decreases with problem size. For the largest problems solved (with $m = 1000$) our PH gives results that, for the test problems considered, are guaranteed to deviate from optimality by no more than one-tenth of one percent. Such closeness to optimality shows the quality of the PH presented in this paper.

Judging from the average percentage deviation figures given at the foot of Table 3 percentage deviation does not seem to vary significantly with problem type. Hence we can conclude that with regard to deviation from optimality our PH is not sensitive to problem type. With regard to computation time it is clear that computation time increases with problem type. Recall from the discussion with regard to test problem generation above that the principal difference between problem types is the number of small (class 4) pieces, 40% for type I, 55% for type II and 70% for type III. Performing a non-linear least-squares regression in order to produce the best-fit equation for our PH of the form: time required $= \phi M^{\gamma}$ (percentage of small pieces)$^{\psi}$ yields $\phi = 0.0025$, $\gamma = 1.42$ and $\psi = 0.67$ (produced using Microsoft Excel and Solver). Hence the observed computational time complexity of our PH is $O(M^{1.42}$ (percentage of small pieces)$^{0.67})$.

5.5. Comparison with optimal algorithms

Of the optimal algorithms for the constrained two-dimensional non-guillotine cutting problem considered in our literature survey above the most effective is the algorithm of Fekete and Schepers [22,23]. We shall therefore in this section compare the results from our heuristic with those of Fekete and Schepers with respect to:

(a) the publicly available test problems shown in Table 1 for which optimal solutions are known; and
(b) the larger problems considered above.

Comparing the results shown in Table 1 with those of Fekete and Schepers [22,23] requires use of Dongarra [19], together with technical information from the Silicon Graphics web site (http://www.sgi.com/) and the SPEC benchmark web site (http://www.spec.org/), in order to get an estimate of the relative speed of the different computers used. For the eight test problems in Table 1 for which the optimal solution was not discovered our best estimate is that, on our Silicon Graphics O2 workstation, Fekete and Schepers [22,23] would require an average computation time of 54 seconds to solve these problems to optimality. By comparison we have an average percentage deviation from optimality of 3.26% with an average computation time of 22.36 seconds.

Note here that Table 1 illustrates that our population heuristic exhibits the same generic computational characteristics as many heuristic algorithms (for many combinatorial optimisation problems) presented previously in the literature, specifically:

(a) generically a "good" heuristic will find optimal solutions for small-sized problems; but
(b) as problem size increases, it will find solutions which, whilst not optimal, are close to optimal in reasonable computation times—certainly in less computation time than that required by an optimal approach.

Referring to Table 1 we have that:

(a) the smaller problems (13 smaller problems out of 21 problems) are solved to optimality;
(b) for the larger problems results are obtained that are 3.26% from optimality in just less than half (more precisely $100(22.36/54) = 41\%$) of the computation time taken by the approach of Fekete and Schepers.

Given that we have generated our larger test problems in the same manner as Fekete and Schepers it is legitimate to ask how our results compare against theirs. Fekete and Schepers consider only very small

Table 4
Fekete and Schepers [23] results

m	Q^*	M	Type I			Type II			Type III		
			Number solved	Average number of tree nodes	Average time (seconds)	Number solved	Average number of tree nodes	Average time (seconds)	Number solved	Average number of tree nodes	Average time (seconds)
20	1	20	10	8	0.56	10	305	2.18	10	299	1.48
30	1	30	10	158	4.48	10	2873	10.64	10	10588	17.67
40	1	40	9	431	22.02	8	14910	51.12	9	62065	103.10
20	3	60	9	287	2.35	7	237144	95.44	8	345130	191.98
20	4	80	8	147864	62.46	6	168530	112.89	2	85729	34.52
Average			9.2	25892	16.73	8.2	68835	45.92	7.8	92306	69.85

problems by comparison to those shown in Table 3. In order to provide some insight into how our heuristic compares against the exact (optimal) procedure of Fekete and Schepers we show in Table 4 summary results taken from [23]. For each (m, Q^*) pair considered Fekete and Schepers attempted to solve to optimality ten test problems. In Table 4 we show:

- the number of problems solved to optimality within a time limit of 1000 seconds on a Sun workstation with a 175 MHz UltraSPARC processor,
- the average number of tree nodes and the average time (in seconds) for those problems that were solved to optimality.

In addition the final row in Table 4 gives the average number of problems solved for types I–III as well as (weighted) averages for the number of tree nodes and solution time.

Considering Table 4 it is clear that as problem size (M) increases the number of tree nodes for the problems that are solved increases rapidly. This is reflected in the increase in computation time as well as an increase in the number of problems that remain unsolved within the 1000 seconds time limit. It is clear that problems of the size shown in Table 4 represent the effective computational limit for the Fekete and Schepers tree search procedure. Indeed Fekete and Schepers themselves state [23] that instances with $M = 80$ seem to represent the limit of their algorithm.

Compare now the size of problem solved in Table 4 (problems up to $M = 80$) with the size of problem solved in Table 3 (problems up to $M = 4000$). It is clear that our heuristic can solve problems that are fifty times larger than those which can be solved using the Fekete and Schepers approach. Whilst our algorithm is a heuristic the percentage deviation figures given in Table 3 indicate that as problem size increases our heuristic performs better, getting results that are (empirically) closer to optimality. Indeed for the largest problems solved our results are very close to optimality.

In summary then we would conclude here that whilst the Fekete and Schepers procedure is an excellent algorithm for small problems it simply cannot deal computationally with problems of the size that can be routinely tackled using our population heuristic.

5.6. Comparison with heuristic algorithms

Recall from our literature survey above that there were four heuristics [29–31,39] for the constrained two-dimensional non-guillotine cutting problem presented previously in the literature. Computational comparison between our PH and these heuristics is difficult principally because:

(a) the authors of these papers have failed to include in their computational work the standard test problems that have been publicly available since 1990 via OR-Library [6] (problems 1–12 in Table 1); and
(b) different papers use different test problems. For example Lai and Chan use test problems in their paper [29] that are different from the test problems used in their paper [30] (both papers were published in 1997).

Moreover note here that as discussed in our literature survey above the largest problem previously solved computationally by any heuristic algorithm in the literature had $m = 97$. By contrast we solve problems up to $m = 1000$ in this paper and have made our test problems publicly available for use by future workers.

However there are two comparisons that can be made between the heuristics published previously and our PH that demonstrate the advantages of our PH. These relate to:

(a) impossible cutting patterns; and
(b) the general case.

We deal with each of these in turn.

Recall from our literature survey above that three of the four heuristics published previously, namely [29–31], used a representation involving permutations (ordered lists) of $\{1, 2, \ldots, M\}$, with the permutation indicating the order in which pieces are cut. Leung et al. [31] noted that there exist problems such that it is *impossible* to produce the optimal cutting pattern from any permutation using a standard bottom-left algorithm or using the difference process algorithm of Lai and Chan [29,30]. Fig. 7 (from [31]) shows two such (trim loss) problems, the first cutting pattern shown there cannot be produced using the bottom-left algorithm and the second cutting pattern shown there cannot be produced using the difference process algorithm.

Because our PH uses a representation that directly addresses the coordinates of any cut piece it is clear that it must be capable of producing all cutting patterns—i.e. there are no impossible cutting patterns for our heuristic in marked contrast to previous heuristics presented in the literature. As an illustration of this our PH finds the two cutting patterns shown in Fig. 7 in 0.01 and 0.03 seconds respectively.

Recall from our literature survey above that all of the four heuristics published previously only dealt with special cases of the general two-dimensional non-guillotine cutting problem ([29–31] dealt with $P_i = 0\ \forall i$ and [39] dealt with $P_i = Q_i\ \forall i$). By contrast the heuristic presented in this paper deals with the general case. To illustrate this we took all of the test problems shown in Table 1 (for which $P_i = 0\ \forall i$) and amended them so that $P_i = 1$ for some pieces i. Specifically we:

(a) set $P_i = 0\ \forall i$ initially
(b) considered each piece $i (i = 1, \ldots, m)$ in turn and for each such piece if

$$\sum_{j=1,\ j\neq i}^{m} (L_j W_j) P_j + L_i W_i \leqslant (L_0 W_0)/3 \text{ set } P_i = 1.$$

Essentially this means that we set $P_i = 1$ for some pieces i whose areas (in total) do not exceed one-third of the area of the stock rectangle. All of these test problems are made publicly available for use by future workers (see http://mscmga.ms.ic.ac.uk/jeb/orlib/ngcutinfo.html). The results can be seen in Table 5. In that table we show:

- an upper bound on the optimal solution taken from Table 1 and corresponding to the case $P_i = 0\ \forall i$;
- the best solution found by our PH and the total time taken (in seconds).

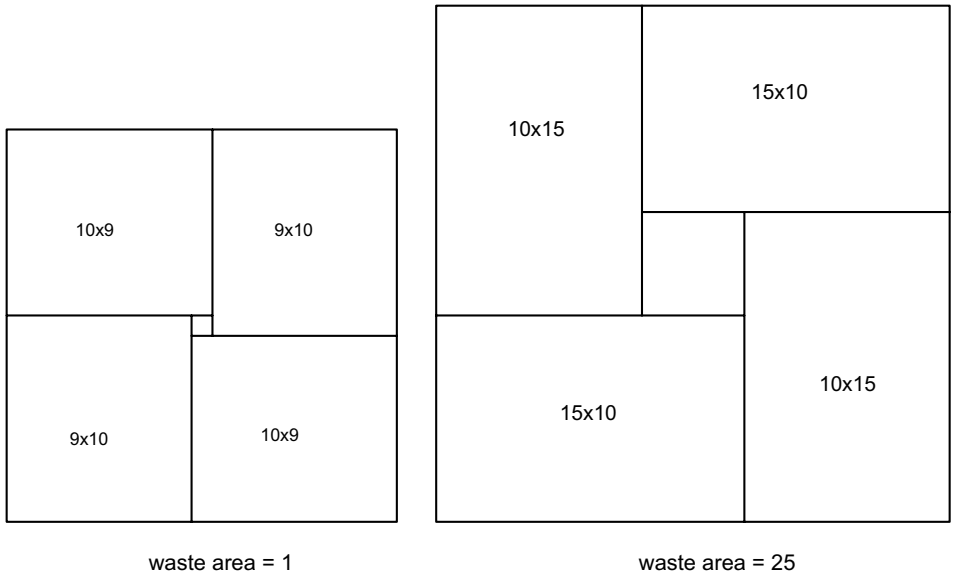

Fig. 7. Impossible cutting patterns.

Table 5
Computational results—general case

Problem source	Problem instance	Problem size			Upper bound	Best solution value	Time (seconds)
		(L_0, W_0)	m	M			
Beasley [5]	1	(10,10)	5	10	164	164	0.02
	2		7	17	230	225	5.53
	3		10	21	247	220	7.85
	4	(15,10)	5	7	268	268	0.01
	5		7	14	358	301	5.05
	6		10	15	289	265	6.81
	7	(20,20)	5	8	430	430	0.01
	8		7	13	834	819	6.54
	9		10	18	924	924	5.64
	10	(30,30)	5	13	1452	n/f	2.38
	11		7	15	1688	1505	2.96
	12		10	22	1865	1666	3.78
Hadjiconstantinou and Christofides [24]	3	(30,30)	7	7	1178	1178	0.25
	11		15	15	1270	1216	2.60
Wang [38]		(70,40)	20	42	2726	2499	6.36
Christofides and Whitlock [15]	3	(40,70)	20	62	1860	1600	6.81
Fekete and Schepers [22,23]	1	(100,100)	15	50	27718	25373	11.86
	2		30	30	22502	17789	5.80
	3		30	30	24019	n/f	4.03
	4		33	61	32893	27556	20.42
	5		29	97	27923	21977	18.41

Note: n/f means that no feasible solution was found.

Note here that for two problems in Table 5 no feasible solution was found by our PH. We examined the data for these two problems and it became clear that no feasible solutions could exist. The reason was that our procedure for problem generation given above (which was purely area based) had picked two pieces which had to be cut where one piece was of length L_0 and the other piece was of width W_0. A moment's thought will reveal that it is impossible to cut two pieces of these sizes. Amending our problem generation procedure to avoid such cases could be easily done but would, in our view, add little value to the results already presented in Table 5. Developing explicit tests to identify when no feasible cutting pattern exists we considered to be beyond the scope of this paper.

6. Conclusions

In this paper we have presented a new non-linear formulation of the constrained two-dimensional non-guillotine cutting problem. Based upon this formulation we presented a population heuristic for the problem. Computational results were presented for a number of standard test problems taken from the literature as well as for a number of large randomly generated problems. We believe that the contribution of this paper to the literature is:

- to present a new, non-linear, formulation of the constrained two-dimensional non-guillotine cutting problem;
- to present an innovative population heuristic based upon this formulation;
- to present computational results for test problems much larger than those considered previously in the literature and to make these test problems publicly available for future workers;
- to demonstrate empirically that as problem size increases our heuristic gets results that are closer to optimality, where for the largest problems solved our heuristic results deviate from optimality by no more than one-tenth of one percent;
- to demonstrate computationally that our heuristic can routinely solve problems that cannot be tackled using the best optimal procedure currently known;
- to illustrate that whereas there exist problems for which previous heuristics can never find the optimal solution the heuristic presented in this paper has no such limitations;
- to present computational results for general two-dimensional non-guillotine cutting, in contrast to previous heuristics presented in the literature which only dealt with special cases.

References

[1] A. Amaral, A.N. Letchford, Improved upper bounds for a two-dimensional cutting problem, Working paper available from the second author at Department of Management Science, Management School, Lancaster University, Lancaster LA1 4YW, England, 1999.
[2] A. Amaral, A.N. Letchford, Comment on an exact algorithm for general, orthogonal, two-dimensional knapsack problems, Working paper available from the second author at Department of Management Science, Management School, Lancaster University, Lancaster LA1 4YW, England, 1999.
[3] M. Arenales, R. Morabito, An AND/OR-graph approach to the solution of two-dimensional non-guillotine cutting problems, European Journal of Operational Research 84 (1995) 599–617.
[4] T. Bäck, D.B. Fogel, Z. Michalewicz (Eds.), Handbook of Evolutionary Computation, Oxford University Press, 1997.
[5] J.E. Beasley, An exact two-dimensional non-guillotine cutting tree search procedure, Operations Research 33 (1985) 49–64.
[6] J.E. Beasley, OR-Library: Distributing test problems by electronic mail, Journal of the Operational Research Society 41 (1990) 1069–1072.
[7] J.E. Beasley, Obtaining test problems via Internet, Journal of Global Optimization 8 (1996) 429–433.

[8] J.E. Beasley, Population heuristics, in: P.M. Pardalos, M.G.C. Resende (Eds.), Handbook of Applied Optimization, Oxford University Press, 2002, pp. 138–157.
[9] J.E. Beasley, P.C. Chu, A genetic algorithm for the set covering problem, European Journal of Operational Research 94 (1996) 392–404.
[10] J.E. Beasley, N. Meade, T.-J. Chang, An evolutionary heuristic for the index tracking problem, European Journal of Operational Research 148 (3) (2003) 621–643.
[11] J.E. Beasley, J. Sonander, P. Havelock, Scheduling aircraft landings at London Heathrow using a population heuristic, Journal of the Operational Research Society 52 (2001) 483–493.
[12] T.-J. Chang, N. Meade, J.E. Beasley, Y.M. Sharaiha, Heuristics for cardinality constrained portfolio optimisation, Computers and Operations Research 27 (2000) 1271–1302.
[13] B. Chazelle, The bottom-left bin-packing heuristic: An efficient implementation, IEEE Transactions on Computers C 32 (1983) 697–707.
[14] N. Christofides, Optimal cutting of two-dimensional rectangular plates, CAD74 Proceedings, 1974, pp. 1–10.
[15] N. Christofides, C. Whitlock, An algorithm for two-dimensional cutting problems, Operations Research 25 (1977) 30–44.
[16] P.C. Chu, J.E. Beasley, A genetic algorithm for the generalised assignment problem, Computers and Operations Research 24 (1997) 17–23.
[17] P.C. Chu, J.E. Beasley, A genetic algorithm for the multidimensional knapsack problem, Journal of Heuristics 4 (1998) 63–86.
[18] P.C. Chu, J.E. Beasley, Constraint handling in genetic algorithms: The set partitioning problem, Journal of Heuristics 4 (1998) 323–357.
[19] J.J. Dongarra, Performance of various computers using standard linear equations software, Working paper (2001) available from the author at Computer Science Department, University of Tennessee, Knoxville, TN 37996-1301, USA. Also available at http://www.netlib.org/benchmark/performance.ps.
[20] K.A. Dowsland, W.B. Dowsland, Packing problems, European Journal of Operational Research 56 (1992) 2–14.
[21] H. Dyckhoff, A typology of cutting and packing problems, European Journal of Operational Research 44 (1990) 145–159.
[22] S.P. Fekete, J. Schepers, A new exact algorithm for general orthogonal d-dimensional knapsack problems, Springer Lecture Notes in Computer Science 1284 (1997) 144–156.
[23] S.P. Fekete, J. Schepers, On more-dimensional packing III: Exact algorithms, Report no. 97.290 available from the first author at Department of Mathematics, Technical University Berlin, D-10623 Berlin, Germany (1997), revised (2000).
[24] E. Hadjiconstantinou, N. Christofides, An exact algorithm for general, orthogonal, two-dimensional knapsack problems, European Journal of Operational Research 83 (1995) 39–56.
[25] R.W. Haessler, P.E. Sweeney, Cutting stock problems and solution procedures, European Journal of Operational Research 54 (1991) 141–150.
[26] P. Healy, M. Creavin, A. Kuusik, An optimal algorithm for rectangle placement, Operations Research Letters 24 (1999) 73–80.
[27] J.H. Holland, Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, University of Michigan Press, 1975.
[28] S. Jakobs, On genetic algorithms for the packing of polygons, European Journal of Operational Research 88 (1996) 165–181.
[29] K.K. Lai, J.W.M. Chan, Developing a simulated annealing algorithm for the cutting stock problem, Computers and Industrial Engineering 32 (1997) 115–127.
[30] K.K. Lai, W.M. Chan, An evolutionary algorithm for the rectangular cutting stock problem, International Journal of Industrial Engineering 4 (1997) 130–139.
[31] T.W. Leung, C.H. Yung, M.D. Troutt, Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem, Computers and Industrial Engineering 40 (2001) 201–214.
[32] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1996.
[33] C.R. Reeves, Genetic algorithms, in: C.R. Reeves (Ed.), Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scientific Publications, Oxford, 1993, pp. 151–196.
[34] C.R. Reeves, Genetic algorithms for the operations researcher, INFORMS Journal on Computing 9 (1997) 231–250.
[35] G. Scheithauer, J. Terno, Modeling of packing problems, Optimization 28 (1993) 63–84.
[36] P.E. Sweeney, E.R. Paternoster, Cutting and packing problems: A categorized, application-orientated research bibliography, Journal of the Operational Research Society 43 (1992) 691–706.
[37] R.D. Tsai, E.M. Malstrom, H.D. Meeks, A two-dimensional palletizing procedure for warehouse loading operations, IIE Transactions 20 (1988) 418–425.
[38] P.Y. Wang, Two algorithms for constrained two-dimensional cutting stock problems, Operations Research 31 (1983) 573–586.
[39] Y.-L. Wu, W. Huang, S.-C. Lau, C.K. Wong, G.H. Young, An effective quasi-human based heuristic for solving the rectangle packing problem, European Journal of Operational Research 141 (2002) 341–358.