Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 168 (2006) 390–402
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

Discrete Optimization

Using tree search bounds to enhance a genetic algorithm approach to two rectangle packing problems

Kathryn A. Dowsland [a,b,*], Edward A. Herbert [c], Graham Kendall [a], Edmund Burke [a]

[a] The School of Computer Science & IT, University of Nottingham, Wollaton Road, Nottingham, NG8 1BB, UK
[b] Gower Optimal Algorithms Ltd, 5 Whitestone Lane, Newton, Swansea SA3 4UH, UK
[c] European Business Management School, University of Wales, Swansea, UK

Received 3 December 2002; accepted 15 April 2004
Available online 25 June 2004

Abstract

A popular approach when using a genetic algorithm in the solution of constrained problems is to exploit problem specific information by representing individuals as ordered lists. A construction heuristic is then often used as a decoder to produce a solution from each ordering. In such a situation further information is often available in the form of bounds on the partial solutions. This paper uses two two-dimensional packing problems to illustrate how this information can be incorporated into the genetic operators to improve the overall performance of the search. Our objective is to use the packing problems as a vehicle for investigating a series of modifications of the genetic algorithm approach based on information from bounds on partial solutions.

Keywords: Genetic Algorithms; Branch and Bound; Cutting; Packing

1. Introduction

The last decade has seen considerable interest in the application of genetic algorithms (GAs) to many different classes of difficult optimisation problems. For example, see Pardalos and Resende, 2002 and Glover and Kochenberger, 2003. The field of cutting and packing is no exception. Such problems tend to be highly constrained in that pieces must be cut or packed within a confined space without overlap. If a packing or cutting pattern is defined by the positions of the pieces it contains, then it is easy to see that any layout

* Corresponding author. Fax: +44 2077 486130.
E-mail addresses: kad@cs.nott.ac.uk (K.A. Dowsland), gxk@cs.nott.ac.uk (G. Kendall), ekb@cs.nott.ac.uk (E. Burke).

0377-2217/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2004.04.030

produced by combining the features of two good feasible layouts is likely to contain overlapping pieces or large gaps. However, genetic algorithms are based on the premise that good partial solutions can be combined via the crossover operator to produce child solutions that are likely to be of high quality. Problems where this is not the case are defined as being highly epistatic, and it is well known that genetic algorithms struggle on epistatic domains (Falkenauer, 1998, p. 34). Examples of researchers recognising these problems are common in many domains. Erben (2000), uses a grouping genetic algorithm (Falkenauer, 1998) to solve a graph colouring problem and successfully applied it to *hard-to-colour* graph examples as well as real-world timetabling problems. Mori and Tanaka (2002) also used a grouping genetic algorithm in designing conference programs and reports good results.

In an early paper Davis (1985) uses a rectangle packing problem to illustrate the difficulty in using a standard GA and suggests an alternative decoding in order to overcome it. This takes the form of an indirect representation, in which individuals are represented as permutations of the pieces. A placement heuristic is used to decode the representation by packing the pieces in the order given by the permutation and crossover involves copying one parent up to the crossover point and then completing the child by adding the missing values in the order they appear in the second parent. Aickelin and Dowsland (2004) used this approach for nurse scheduling and Ozcan and Alkan (2002) applied a similar method to university course timetabling. This approach remains popular for cutting and packing problems (Prosser (1988), Dighe and Jakiela (1995), Liu and Teng (1999).

Although these implementations work reasonably well, intuitively there are two drawbacks. First, only that part of the string before the crossover point is preserved in its entirety. Thus the information inherited from the second parent is less than that inherited from the first as, to maintain feasibility, the second parents genes are not kept together as the genes already inherited from the first parent must be deleted. Other crossover operators could be used (for example, partially matched crossover (PMX) (Goldberg, 1989). However, they too have problems. For example, after applying the PMX operator the offsprings could decode to solutions which differ widely from their parents. In (Falkenauer, 1998, p. 90), an example of a bin packing problem is given and he states *''In other words, the PMX transmits information which more often than not acquires a different meaning in the new chromosome''*. Second, even when all the information is preserved (for example crossing over ABCDEF and BACEFD at position 3) only the first part of the string will preserve its meaning in terms of the physical arrangement of the pieces. The positions of *E*, *F* and *D* will depend on the layout of the pieces preceding them. Thus, even if the information from both parents is preserved in the genotype only the first part of the string will preserve its meaning in the phenotype. The result of these two observations is likely to be a large variation in the fitness values of the schema left undisturbed by crossover, thus affecting the efficiency of the GA. The motivation for the work in this paper is to exploit additional information inherent in the front section of the strings, in an attempt to overcome the deficiencies caused by lost information in the tails. This information is derived from a more traditional approach to discrete optimisation problems, that of tree search.

The application of tree search methods, often based on branch and bound, to cutting and packing problems have been widely reported in the literature. In many cases moderately sized problems have been solved efficiently by these means. Dowsland (1987) suggested a branch and bound approach for the pallet loading problem, in which the objective is to maximise the number of copies of a small rectangle that can be fitted without overlap into a larger rectangle. Beasley (1985a) successfully solved the problem of maximising the value of different sized pieces cut from a fixed area using a mixture of geometric bounds and Lagrangian relaxation, while Christofides and Whitlock (1977) solved the constrained guillotine cutting problem by branching on the cutting positions and using the easier unconstrained problem to generate bounds. In the field of irregular cutting, the heuristic of Adomowicz and Albano (1976) is based on the greedy selection of branches in the

search tree of feasible solutions. More recently Fekete and Schepers (1997, 2004a,b) have presented a branch and bound approach that uses graph theoretic results to make a significant reduction in the size of the search tree, which improves on the above approaches in terms of the size of problem that can be solved to optimality within a reasonable time. Here we exploit the relationship between the branches of search tree and the strings in an indirect GA representation to incorporate bound information into the GA. Two classes of rectangle packing problems are used to provide empirical evidence of the improvement in performance that can be obtained by using bound-based information in this way.

It should be emphasized that our objective is simply to use these packing problems as a vehicle for investigating the effectiveness of incorporating bound based information into the GA and not necessarily to compete with the best two dimensional packing algorithms currently available. For this reason we only include references to the literature on packing problems where they are directly relevant. A recent survey of approaches to two-dimensional packing problems can be found in Lodi et al. (2002). However we note that in spite of the success of the exact approach of Fekete and Schepes there is still considerable interest in heuristic approaches to the problem, for example Lodi et al. (1999), and Faroe et al. (2003).

2. The relationship between indirect GA representations and tree searches

The relationship between the representation of individuals in the GA and a tree search is derived by mapping the string positions to the levels of the tree, and the value of each position in the string to the choice of branch at that level. Thus a binary coding of n bits would be represented as a binary tree of depth n and each individual would correspond to a path from the top of the tree to a terminal node (with a 0 representing the left-hand branch and a 1 the right-hand branch at each level). Similar mappings can be derived for permutations and other types of strings. If it is possible to calculate bounds at each node in the tree, then we have additional information about the partial solutions represented by positions 1 to k in each of the individual strings. If the GA is based on an indirect coding in which the individuals represent an ordering, and the decoder builds solutions by acting on the entities involved in the given order, then such an analogy is a particularly intuitive one.

The idea of combining branch and bound with a GA is not completely new. For example Cotta et al. (1995) incorporate a tree search into their crossover operator for the travelling salesman problem in order to select the best way of combining material from both parents. Of more interest from our point of view is the work of Nagar et al. (1995) and Tamura et al. (1994). Nagar et al. investigate the use of a mutation operator to destroy partial solutions corresponding to paths to bounded nodes in the search tree. They describe a GA solution to a two-machine flowshop scheduling problem in which individual solutions are represented as permutations of jobs and the objective is to minimise a weighted sum of makespan and average completion time. Prior to running the GA they execute a branch and bound tree search down to a predetermined depth, k (where k is chosen according to the run-time available) and suitable bounds are calculated and recorded at each node. During the execution of the GA the partial solutions up to position k are mapped onto the correct node. If the lower bounds indicate that no path below this node can lead to an optimal solution the string is subjected to a mutation operator that has been specifically designed to change the early part of the string in a favourable way. The drawback of this approach is the pre-processing time required to search the upper regions of the tree. Tamura et al. (1994) take an alternative approach. They tackle a job-shop scheduling problem and start from an IP formulation. For each variable they take the range of possible values and partition this into a set of sub-ranges, which are then indexed. The strings in the GA are then defined so that each position represents a variable, and its value corresponds to the index of one of the subranges. The fitness of a string is calculated using Lagrangian relaxation to obtain a bound on the optimal solution subject to the constraints

that the values of the variables fall within the correct ranges. When the GA terminates, an exhaustive search of the region identified as the most promising is carried out to produce the final solution.

These two approaches differ in their consequences if the bound information is misleading. The latter requires that the bounds provide a good measure of the quality of the final solution in order to ensure that the right region is highlighted for further investigation. The former approach is more robust, in that, if the bounds are slack the algorithm will simply revert to a conventional GA. In this way the bound information is used to boost the GA when such information is accurate. Our approach is similar. However, although for reasons discussed later we found it necessary to use a modified mutation operator, our primary focus is on the use of bound-based information to modify the crossover operator. We also calculate the bounds as required, thereby dispensing with the need for pre-processing, and more importantly, allowing the incorporation of bound information anywhere on the string, rather than in the first k positions.

In the following sections we will illustrate our approach on two different packing problems and show that it does improve the operation of a standard GA. As with many other suggested improvements to a standard GA our modification results in further decisions for the algorithm designer. We therefore start by describing the operator in detail and highlight some of the potential problems and additional decisions arising from its use.

3. The GA and bounded crossover operator

Our experiments are based on a generational (as opposed to steady-state) GA in which a fixed proportion of the population are reproduced directly into the next generation, with the remaining individuals being produced by crossover. The crossover operator produces two children from two parents using standard one-point crossover in the case of binary strings, or an appropriate modification for permutations or cases where the number of occurrences of each value is constrained. Individuals for reproduction and crossover are chosen according to fitness using roulette wheel selection. As our objective is to measure the effect of the bounded crossover operator we chose not to use an elitist strategy. The parameters governing the search (i.e. the population size, number of generations, crossover and mutation rates) differ according to the different types of problem tackled.

The new, bound-based crossover is a simple modification to one-point, or any other crossover that preserves that portion of the parent before the crossover point. We assume that the decoder allows us to define partial solutions by decoding from the start of the string to any point along the string, and that we have a target solution value with which to compare the bounds. This may be the known optimal value, or may be based on the best solution found so far. We then calculate a bound point on each individual by working down the string and, assuming a maximisation problem, calculating the upper bound for the partial layout defined by the string up to the current position. If this is lower than the target then the bound point is defined at the current position and the process moves to the next individual. Our modification is based on the following observation. If an individual has a bound at point b, and contributes to the next generation by crossover after point b, the child cannot represent an optimal solution as it also is bounded at point b. It therefore makes sense to ensure that crossover takes place at a point earlier than b. Our modified crossover operator achieves this by limiting the set of potential crossover points to 1 to $b-1$. This raises three issues that need to be resolved.

(1) We have already stated that the crossover operator produces two children from two parents. It is likely that the bound points on these two parents will be different. Thus we need to decide which will be used in order to generate the crossover point. Selecting the later one could mean that one child inherits the whole sequence up to and including the bound point from one of the parents, thus weakening the effect of the special operator. Conversely selecting the earlier one will cut out potentially good solutions obtainable only by crossing over at a later point. A third option is

to generate different crossover points for each child, based on the bound point of the parent contributing to its front end. This has the potential disadvantage that information between the crossover points in one of the parents will be lost. For example if we have parent $A=(a,a,a,a,a,a,a,\ldots)$ bounded at position 3 and $B=(b,b,b,b,b,b,b,\ldots)$ bounded at position 6, if we generate the crossover point between 1 and 5 children such as $(a,a,a,a,b,b,b\ldots)$ may be generated. This child would also be bounded at 3 and thus cannot be optimal. Conversely if we only allow crossover up to point 3, children such as $(b,b,b,b,b,a,a,\ldots)$, which might be optimal, will not be generated. Finally if we generate two separate crossover points, say at positions 2 and 5 the children will be $(a,a,b,b,b,b,b,\ldots)$ and $(b,b,b,b,b,a,a\ldots)$. Thus the material from parent A in positions 3–5 does not contribute to either child. As none of these options is without its disadvantages we consider all three possibilities in our initial experiments.

(2) As the problems being considered are inherently difficult and the initial population is produced entirely at random it is likely that, in early generations, the bound points will fall early in the strings. Because the effects of recombination are limited to the portion of the string before the bound point the tails of the strings will only be affected by the conventional background mutation operator. Nevertheless they will still be subjected to fitness-based selection, and may therefore be prone to premature convergence. It may, therefore, be necessary to employ additional mechanisms to ensure sufficient diversity is maintained beyond the bound point. Possible approaches would be the use of additional mutation operators, or a second crossover point.

(3) The final issue is related to 2 above. As the main genetic operator is not operating on the whole string, this raises the question as to whether we should include the whole string in the fitness measure, or simply measure fitness up to the bound point. Material beyond the bound point is inherited by one of the children. This suggests that it should contribute to fitness. However, as pointed out earlier, information beyond the crossover point is likely to be less meaningful as it corresponds to different arrangements of pieces depending on the earlier part of the string. As we are also suggesting that it may be necessary to subject the latter part of the string to a more aggressive mutation the argument for not incorporating information beyond the bound-point in the fitness function is equally compelling. Both options are therefore addressed in the experimentation.

In the following two sections we outline the implementation of our basic GA and its bounded modification to two different packing problems and use empirical tests to judge the influence of the modification and to select the best approach to the problems outlined above. The first problem is chosen as a pilot study for its simplicity, while the second is chosen to test the potential of the method on a more complex problem.

4. Identical pieces

The problem selected for our initial investigations is the pallet loading problem of maximising the number of copies of a small rectangle (the box) that can be packed orthogonally without overlap into a larger containing rectangle (the pallet). This problem was chosen because it is relatively simple and has been well solved by other methods. This not only has the advantage that we can compare the solutions from a GA against a known optimum, but that we can use this information as the target value, thus enabling us to measure the effect of the bounds more accurately. The simplicity of this problem provided an initial platform for testing our basic ideas. In addition, because the problem can be regarded as a problem of two box types—one for each orientation—without any restriction on the numbers of each type, we can use binary strings and one-point crossover in our basic implementation. Finally good bounds based on the amount of waste in partial packings were readily available as a result of the work described in Dowsland (1987). The objective of these experiments was to determine whether or not the bound-based crossover was worth pursuing and allow us to identify, and hopefully correct, any potential pitfalls before applying the procedure to more complex problems.

Table 1
GA parameters for pallet loading experiments

Population size	20
No. generations	100 (max)
Crossover rate	60%
Mutation rate	0.05

In order to implement a standard GA and compare its results with those of a bounded crossover operator we need to make a number of decisions. The basic parameters were decided as a result of empirical tests and are given in Table 1. The representation and decoder, the way in which we define/calculate the bounds and the target value, the fitness functions, crossover operators and mutation operators used are dealt with in the following subsections.

4.1. Representation and decoders

As the problem is concerned with identical boxes that can be placed in one of two orientations solutions are coded as binary strings. Because the number of boxes in an optimal layout is known the string length can be set to equal this number. Davis's (1985) experiences with his indirect representation for different sized pieces indicated that solution quality was sensitive to the decoder. We therefore consider three different decoders in our experiments. For the first two a *0* represents a box in the horizontal orientation and a *1* a box in the vertical orientation. The boxes are then taken in order and placed on the pallet according to a placement policy. The first decoder (BL) operates a bottom left policy in which the height of the lowest feasible position for the piece is determined and it is then placed at the leftmost position at this height. The second decoder (DROP) is based on three movements. It starts by 'dropping' a piece into the pallet by allowing it to slide down against the right-hand edge until it is blocked by another piece or by the bottom edge of the pallet. It is then moved as far as possible to the left, keeping its y-coordinate constant. Finally, if it is not already resting on the piece immediately below it is dropped as far as possible keeping the x-coordinate constant. The final decoder (COORD) is based on the tree search of Dowsland (1987). Here the feasible placement positions are reduced to those that are an integral combination of piece lengths and widths from the nearest pallet edges in a horizontal and vertical direction, and are then ordered starting at the bottom left and working along the pallet in rows. Horizontal positions are ordered before vertical positions at the same co-ordinates. These ordered positions then define the branches of the tree, with the feasible branches from each node representing the positions that are still feasible. As the number of remaining feasible positions depends on the partial layout already in place, the number of branches will differ at different nodes at the same level. However, only branches representing positions that would overlap with the current placement will be precluded at the next level. Therefore in practice only the first few branches need to be considered. For the purposes of this study we reduce this to the first two branches. At the first few nodes these will correspond to horizontal and vertical placements at the same position, lying next to the previously placed box. However, later in the tree two branches may correspond to pieces of the same orientation in different positions. Although this decoder will not explore all the branches defined by the original tree search, we were unable to find a counter-example where all optimal solutions were eliminated by this policy. Similar comments apply to the BL and DROP decoders. Moreover we were able to show that all the decoders can produce at least one optimal solution on all the data sets used in our tests.

4.2. Bounds and targets

All the decoders produce partial layouts with the pieces filling up the pallet in rows from the bottom left. Thus any gaps left 'behind' the current packing can be identified as waste. As the number of boxes in an optimal solution (N_{opt}) is known, the upper bound on the total wasted area in an optimal solution is given by Pallet_Area $-$ N_{opt}* Box_Area. Dowsland defines a number of different ways of defining bounds based on the waste. These can be broadly partitioned into three classes:

1. *Covered waste*. Waste lying directly beneath the bottom edge of a currently packed box.

2. *Line waste*. Unusable waste lying behind (i.e. to the left of) a currently placed piece. This typically takes the form of a gap narrower than the width of a piece, but depending on the decoder other broader definitions are possible.
3. *Induced waste*. Waste that has not yet a physical space in the layout, but that must be generated later. For example if the distance from the rightmost or topmost edge of the partial layout to the edge of the pallet is not an exact combination of box dimensions.

These three definitions can be combined in such a way as to avoid double counting. The total waste can then be accumulated and recorded as the decoder adds each box to the partial layout. We know the value of the optimal solution so this can be used as the target. This means that the point at which the accumulated waste exceeds that in an optimal solution can be defined as the bound point for the string.

4.3. Crossover

Standard one-point cross-over will produce feasible strings that have the same physical meaning before the crossover point in the children, for all three decoders. This was therefore used as the basis for the modified crossover. Three variants, denoted E_{BP}, L_{BP} and O_{BP} and defined as follows, were used in the experiments. Let b_1 be the position of the bound point on the first parent and b_2 that on the second. E_{BP} is based on the earlier bound-point i.e. the crossover point is a random point less than min (b_1, b_2). L_{BP} is based on the later bound point i.e. the crossover point is a random point less than max (b_1,b_2). O_{BP} (denoting own bound point) uses two different crossover points with the crossover point for the first child being generated before b_1 and that for the second child before b_2.

4.4. Fitness

The natural definition of fitness is the number of boxes packed before the top of the pallet is breached. In our experiments we denote this by f_1. The corresponding measure of fitness up to the bound-point is therefore the number of boxes packed up to that point i.e. the position of the bound-point itself. This is denoted by f_2. We also define a combined fitness given by $(f_1+f_2)/2$. Finally, as we expect the difference between the fittest and least fit solutions to be small, thereby resulting in relatively flat fitness functions, we also ran an experiment using completely random selection. We denote the randomly selecting function by f_0. Of course, in this case, we are not generating a measure of fitness but, rather, we are saying that fitness dos not play a role in the selection procedure.

4.5. Mutation

Our basic mutation operator was the standard one of flipping each bit with a known probability. However, as outlined in the previous section, we were concerned that premature convergence in the tails of the strings may require a more aggressive mutation operator. Two forms of additional mutation, denoted SPLIT1 and SPLIT2, were tried. SPLIT1 simply applies a higher probability of mutation after the bound-point. SPLIT2 is based on the analogy with a tree search where, when a node is fathomed, the search tries the next option from the node at the previous level. This behaviour is mimicked here by mutating the gene at the bound-point with a given probability.

4.6. The experiments

Experiments were carried out on three different data sets. The first, P1, is a small data set of 10 problems used in Herbert and Dowsland (1996). The second, P2, is a set of 100 randomly generated problems with the ratios *Pallet_area:Box_area*, *Pallet_length:Pallet_width*, *Box_length:box_width* lying in the ranges 20:1 to 30:1, 1:1 to 2:1 and 1:1 to 4:1 respectively. The third, P3, consists of a further 100 problems with the box and pallet aspect ratios in the same range as P2 but with the area ratio in the range 30:1 to 40:1.

In order to gain some initial feel for the performance of the decoders and the way in which the behaviour was effected by the bound-based

information, the GA with standard one-point crossover and each of the three modified crossovers, with each of the four fitness functions, and each of the three decoders was run using 10 random starts on all 10 problems in P1. The results are summarised in Table 2.

The results give rise to the following observations. First the decoder is clearly a significant factor in the quality of the results, with the COORD decoder outperforming the other two throughout. Second, comparison of the results using f_0 (random selection) and f_1 supports the view that f_1 might not differentiate sufficiently between different solutions. f_2 is clearly more successful across the range of other treatments, suggesting that f_2 does exhibit more variety, and that the number of pieces packed up to the bound-point is a good proxy for fitness in this case. However, the most important results from the point-of-view of this study are that the O_{BP} modified cross-over, especially when combined with a fitness function that also takes account of the bound, results in considerable improvements over the basic GA. A closer examination of the data reveals considerable room for improvement. In particular there is one problem in this data set on which every combination of operators fails at least 90% of the time. Although these results are encouraging, 10 problems are clearly not sufficient to draw any general conclusions. The experiment was therefore repeated on set P2—this time allowing just 1 run per problem. These results are given in Table 3. They exhibit the same trends as highlighted on P1 and an r-sample binomial test confirmed that O_{BP} does improve the performance of all three decoders and that the difference is greater for the COORD decoder.

Having established that bound information in the form of bounded crossover, and a bound-based definition of fitness can improve the search, we are now ready to address the potential problem of premature convergence in the tails. Given the quality of the results already obtained for the best option, we used the more challenging data set P3 to test the different mutation operators. The results of repeating the above experiment, but using only the more successful COORD decoder, on P3 are given in the first four columns of Table 4.

The algorithm was then run on this data set with the COORD decoder and fitness function f_2 using a variety of mutation rates. First combinations of a base rate ranging from 0 to 0.04 in steps of 0.01 with a SPLIT1 rate ranging from 0 to 1 in steps of 0.1 were run for all four crossovers. The results suggested that low rates of SPLIT were preferable to higher rates, but that there was little to be gained by this form of additional mutation. Running the same set of experiments with SPLIT2 did, however, produce positive results, showing increased performance with an increased probability of a SPLIT2 mutation, right up to a probability of 1.0. The best results occurred where the background mutation rate was lowered to around 0.02. The result for SPLIT2 with probability 1.0 and background mutation at 0.02 is shown in the final column of Table 4. All these results were confirmed at the 95% significance level using Tukey's test.

These experiments confirm that the use of bound information to drive fitness crossover and mutation can make a significant improvement to an indirect decoder for the pallet loading problem. It is possible that the above results could be improved by tuning the other parameters, increasing

Table 2
Number of optimal terminations (of 100) for each combination of decoder, crossover and fitness on problems P1

Crossover	COORD				BL				DROP			
	f_0	f_1	f_2	f_3	f_0	f_1	f_2	f_3	F_0	f_1	F_2	f_3
OP	79	78	89	87	49	48	59	59	34	39	51	40
E_{BP}	83	78	90	90	51	49	58	53	37	43	47	43
L_{BP}	85	83	89	91	51	49	59	53	39	41	56	41
O_{BP}	87	90	90	91	51	54	63	53	52	45	57	53

Table 3
Number of optimal terminations (of 100) for each combination of decoder, crossover and fitness on problems P2

Crossover	COORD				BL				DROP			
	f_0	f_1	f_2	f_3	f_0	f_1	f_2	f_3	f_0	f_1	f_2	f_3
OP	61	67	88	75	50	51	73	57	38	38	63	47
E_{BP}	58	62	87	69	48	53	70	56	38	40	61	50
L_{BP}	58	69	83	76	54	49	77	64	38	42	71	53
O_{BP}	74	74	90	84	58	63	78	67	47	50	69	61

Table 4
Number of optimal terminations (of 100) for the COORD decoder and each combination of crossover and fitness on problems P3

Crossover	COORD				
	f_0	f_1	f_2	f_3	f_2/s_2
OP	50	60	62	59	74
E_{BP}	50	55	61	56	78
L_{BP}	54	59	71	59	79
O_{BP}	63	61	77	70	90

the population size or including an elitist strategy or rank based selection. However, as our objective simply to test the potential of the approach, we move on to the more challenging problem of packing different sized rectangles.

5. Non-identical pieces

Many different variants of the two dimensional rectangle packing problem exist. Here we tackle the following version:

Given a large containing rectangle and set of different sized smaller rectangles, each with an associated value, and a (small) fixed number of rectangles of each size, maximise the value of the smaller pieces packed within the larger rectangle without overlap. Once again we will consider the different decisions involved in designing and testing a bounded crossover operator for these problems.

5.1. Representation and decoders

This is essentially the same problem as that tackled by Davis (1985). In his case the values of the pieces were considered equal to the area. Following his suggestion the pieces could be indexed and individuals represented as permutations of the pieces. However, this representation is wasteful where there may be several copies of each piece. For this reason we represent individuals as strings with length equal to the total number of pieces, over an alphabet of size equal to the number of piece types, such that the number of occurrences of each value is equal to the number of pieces of the type represented by that value. Our decoders will be placement policies that will act on the pieces in the order they appear in the string. One of these decoders will be our bottom-left decoder. However, in the light of the success of the COORD decoder in the previous section, we consider a second option in which the feasible positions for each piece are defined as those made up of an integral combination of the different piece dimensions from the nearest edge. These are ordered in rows from the bottom-left corner and a piece is placed in the first available feasible position.

5.2. Bounds and targets

The calculation of waste area within the packing can be carried out with only minor modifica-

tion to the routines used for the pallet loading problem. However, as we have chosen the more general objective of maximising value rather than the area of the pieces packed this cannot be used directly to obtain a bound. For any partial layout let the value of the pieces already packed be denoted by V, their area by A, and the waste bound by W. An upper bound on all solutions including this partial layout is given by the knapsack problem:

$$\begin{aligned} \max \quad & V + \sum_{i \in R} v_i x_i \\ \text{s.t.} \quad & \sum_i a_i x_i \leqslant P - A - W, \\ & x_i \leqslant R_i, \end{aligned} \tag{1}$$

where v_i and a_i represent the value and area of piece type i, R_i is the number of pieces of that type remaining, P is the area of the containing rectangle and x_i is an integer variable representing the number of copies of a piece type i fitted into the knapsack. Although this is itself an NP-hard problem the LP relaxation can be solved easily by inspection, by sorting the pieces into v_i/a_i order and allocating each x_i up to its upper bound until the knapsack is filled (see Martello and Toth, 1990).

Beasley (1985a,b) suggests two relaxation bounds based on an IP formulation of the problem. The first is the standard LP relaxation and the second is obtained by relaxing the no-overlap constraints in Lagrangian fashion, using a standard subgradient search to set the multipliers. Finally, as it is unlikely that all the pieces will fit a bound point can be triggered when the next piece will not fit into the containing rectangle. The LP bound is at least as tight as the Lagrangian bound (LR) so there is no merit in calculating both. Tests using standard LP code suggested that the LP bound would be too slow, as it is necessary to calculate the bound at each position on the string for every individual. The LR bound proved less computationally intensive. However initial tests suggested that it provided little improvement over the waste/knapsack bound alone in terms of solution quality, but did incur a far greater cost in terms of computation time. We therefore use the waste/knapsack bound, or set the bound point at the first non-fitting box if this occurs earlier.

In general, the optimal solution for these problems will not be known beforehand. We therefore set the target value to be better than the best solution to date. In order to save computational effort we also include a global upper bound on the solution to allow the search to terminate if a solution of this value is reached. Here we use the tighter of Beasley's LR bound and the above knapsack bound for the case where no pieces have been packed (or the optimal solution for those problem instances where this is known).

5.3. Crossover

Standard one-point crossover will not be feasible for this problem as the numbers of pieces of each type will not be preserved in the children. We therefore use a modification. First a standard one-point crossover is carried out. That part of the child before the crossover point is then preserved, while the second part is altered to correct any over or under-represented pieces as follows. Starting from the tail of the string any over represented pieces are removed. The remaining material is then shifted forwards to fill any gaps. Finally, again starting from the tails, under-represented pieces are added in the order they appear in the other parent. For example if we assume the two parents below with a crossover point after the fifth element

Parent 1	*A*	*B*	*A*	*C*	*C*	*D*	*A*	*D*	*B*	*B*
Parent 2	*B*	*C*	*D*	*B*	*A*	*C*	*B*	*A*	*D*	*A*
Stage 1	*A*	*B*	*A*	*C*	*C*	*C*	*B*	*A*	*D*	*A*
Stage 2	*A*	*B*	*A*	*C*	*C*		*B*	*A*	*D*	
	A	*B*	*A*	*C*	*C*	*B*	*A*	*D*		
Stage 3	*A*	*B*	*A*	*C*	*C*	*B*	*A*	*D*	*B*	*D*

This crossover operator will be referred to as One-point(m), and can now be modified by limiting the range of the crossover point. In view of the lack of success with L_{BP} and E_{BP} for the pallet loading problem we do not include either in our experiments here. Instead we select a hybrid of L_{BP} and O_{BP}, denoted MO_{BP}. This operator generates a crossover point as in L_{BP} according to the

later bound-point of the two parents. If this falls before the earlier bound-point of the two parents it is used for both children. If not a second crossover point is generated before the earlier bound-point and crossover proceeds as for O_{BP}.

5.4. Mutation

As was the case with crossover, a new mutation operator must be introduced in order to preserve feasibility. In this case, the normal mutation operator swaps the values of a randomly selected pair of genes. It is also necessary to consider the introduction of additional mutation operators at or after the bound point. In view of the fact that the tail of the string is somewhat disrupted by the crossover operator, and the lack of influence of the SPLIT1 mutation for the pallet loading problem no such equivalents are considered here. Instead we introduce two new operators. The first, SCRAMBLE, assumes that the tail of the string has been so disrupted as to be unmeaningful and simply randomises the order of all values after the bound point. The second, FLIP, is designed to emulate the SPLIT2 mutation for pallet loading. It swaps the value at the bound point with the value of a randomly selected later gene (element of the chromosome).

5.5. The experiments

These variants were run on the 12 problems examined by Beasley (1985a,b) and available at http://www.ms.ic.ac.uk/info.html, each problem being run through 10 replications of the GA using different random number streams and different starting solutions. The results are shown in Table 5.

Once again the results indicate that COORD is superior to bottom left and that incorporating bound information into the crossover and mutation operators improves on the basic GA with 75% of runs solving optimally when the best combination of bound based crossover, mutation and fitness are employed. When applied in conjunction with the COORD decoder there is little difference between the two bound based mutations—SCRAMBLE and FLIP—in improving on the performance of the modified crossovers. Closer inspection of the running of the algorithms revealed that the knapsack/waste bounds rarely came into play, and that the bound point was usually triggered by a piece failing to fit into the containing rectangle. Under these circumstances the remainder of the string has no influence on fitness and it is therefore not surprising that both mutations give similar results. The same observation explains why the there is little difference between the two fitness functions. As no further boxes are accommodated after the bound point f_1 and f_2 are frequently equal.

This may be a characteristic of the data that would not be as apparent in different or larger problems. Therefore further tests were carried out using a randomly generated data set of 20 problems, each consisting of 50 boxes, of between

Table 5
Percentage optimal terminations each combination of decoder, crossover and fitness on Beasley's 12 problems

Crossover	Mutation	BL decoder		COORD decoder	
		f_1	f_2	f_1	f_2
One-point(m)	Normal	49.2	49.2	55.0	55.0
O_{BP}	Normal	52.5	52.5	55.0	59.2
MO_{BP}	Normal	55.8	55.9	65.0	60.1
One-point(m)	Scramble	68.3	66.6	69.2	70.0
O_{BP}	Scramble	69.2	70.0	70.0	73.3
MO_{BP}	Scramble	69.2	75.8	74.2	70.0
One-point(m)	Flip	65.0	56.7	69.2	71.7
O_{BP}	Flip	64.2	63.3	72.3	70.0
MO_{BP}	Flip	63.3	57.5	70.8	75.0

8 and 12 box types, with up to 10 boxes of any type. As it seems sensible to assume that the efficiency of the bounded algorithm will depend on the tightness of the bounds the problems were generated in 4 groups. In the first group the area of the containing rectangle is 50% of the total piece area, and in the second, third and fourth groups it is 70%, 85%, and 100% respectively. All combinations were again run 10 times on each of the 20 problems. As the optimal solution to these problems is not known it is not possible to present the results in terms of the number of optimal terminations. Instead we carried out multi-factor analysis of variance, followed by a Tukey test to distinguish between those factors with more than one degree of freedom.

The mean solution value for each of these factors over full range of experiments is given in Table 6. The results of the ANOVA test showed that crossover, mutation and the fitness definition were all significant at a 95% significance level. In the case of the mutation operator the Tukey test confirmed that the FLIP mutation is significantly better than the SCRAMBLE mutation, which is itself significantly better than the normal background mutation alone. However, in the case of the crossovers the differences are less clear-cut. The Tukey test confirmed that the bound based crossovers were better than standard one-point crossover, but failed to find a significant difference between the two. An examination of the performance of the different variants on the individual problems highlighted differences between the four groups. In the first three groups, we expect many pieces to remain unfitted and the best recorded solution was found by a combination of a bound based crossover and mutation operator in all but one case where one-point crossover combined with SCRAMBLE produced the best result. For the fourth group, where almost all the pieces can be fitted one-point crossover always found the best solution. However, this performance was always in conjunction with a bound-based mutation and was frequently equalled by the other crossovers. The later observation supports the conjecture that where the bounds are slack the algorithm will tend to revert to the standard GA, and the use of the bound based operators will not be detrimental. Solution times on a Pentium III/500 were also very reasonable ranging from 15 to 39.2 seconds per problem instance.

Table 6
Mean solution quality for different options on randomly generated data with 50 pieces per data set

Feature	Option	Mean
Fitness	f_1	4942.0
	f_2	4946.8
Mutation	Normal	4907.2
	SCRAMBLE	4957.3
	FLIP	4968.8
Crossover	1-point	4930.1
	O_{BP}	4950.0
	MO_{BP}	4953.0

6. Conclusions

This paper has introduced a new way of utilising bound based information to improve the performance of a genetic algorithm. Experiments with two rectangle packing problems have shown that such modifications can be successful when applied to the crossover operator alone, but that further improvements can be obtained when reinforcing the effect in the mutation operator and fitness function. Although these experiments were carried out with one-point crossover they could be extended to other operators such as two-point or PMX by ensuring that the first crossover point is generated before the bound point. Indeed, this may be beneficial in avoiding premature convergence after the bound point. As the objectives of this research were to measure the effect of such operators no attempts were made to improve performance by adjusting the parameters, or introducing features such as elitism that have been shown to be successful for many GA implementations reported in the literature. It is likely that such modifications would improve performance still further. The purpose of this paper was not to study two rectangle packing problems from the point of view of finding the best algorithm for the job. Instead, the main goal of the paper was to use these problems as a vehicle to study genetic algorithm approaches and the effect that

information from bounds on partial solutions might have on those approaches. Although this study has concentrated on two dimensional packing problems, effective bounds exist for many other optimisation problems and it would be interesting to see whether the ideas presented here can improve the performance of indirect genetic algorithms in a wide range of application areas.

References

Adomowicz, M., Albano, A., 1976. Nesting two-dimensional shapes in rectangular modules. Computer Aided Design 8, 27–33.

Aickelin, U., Dowsland, K.A., 2004. An Indirect Genetic Algorithm for a Nurse Scheduling Problem. Computers and Operations Research 31, 761–778.

Beasley, J.E., 1985a. An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 1, 49–64.

Beasley, J.E., 1985b. Bounds for two-dimensional cutting. Journal of The Operational Research Society 36, 49–64.

Christofides, N., Whitlock, C., 1977. An algorithm for two-dimensional cutting problems. Operations Research 25, 30–44.

Cotta, C., Aldana, J.F., Nebro, A.J., Troya, J.M., 1995. Hybridising genetic algorithms with branch and bound techniques for the resolution of the TSP. In: Poras, C.C., et al., (Eds.), Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, France, pp. 277–280.

Davis, L., 1985. Applying adaptive algorithms to epistatic domains. In: Michandani, P.B., Francis, R., (Eds.), Proceedings of the Ninth International Joint Conference on Artificial Intelligence, vol. 1, pp. 162–164.

Dighe, R., Jakiela, M.J., 1995. Solving pattern nesting problems with genetic algorithms employing task decomposition and contact detection. MIT Internal Report.

Dowsland, K.A., 1987. An exact algorithm for the pallet loading problem. European Journal of Operational Research 31, 78–84.

Erben, W., 2000. A grouping genetic algorithm for graph colouring exam timetabling. Burke, E.K., Erben, W. (Eds.), Selected Papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000), LNCS 2079, Springer, Berlin, pp. 132–156.

Faroe, O., Pisinger, D., Zachariasen, M., 2003. Guided local search for the three-dimensional bin packing problem. INFORMS Journal on Computing 15, 267–283.

Falkenauer, E., 1998. Genetic Algorithms and Grouping Problems. Wiley, New York.

Fekete, S.P., Schepes, J., 1997. A new exact algorithm for general orthogonal d-dimensional knapsack problems. In: Algorithms—ESA '97, vol. 1284, Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 144–156.

Fekete, S.P., Schepers, J., 2004a. An exact algorithm for higher-dimensional orthogonal packing. Technical report, Univ. of Cologne, Center for Parallel Computing, (available at http://www.math.tu-bs.de/~fekete/publications.html)—revised and re-submitted to Operations Research.

Fekete, S.P., Schepers, J., 2004b. A combinatorial characterization of higher-dimensional orthogonal packing. Mathematics of Operations Research 29, 353–368.

Glover, F., Kochenberger, G., 2003. Handbook of Metaheuristics. Kluwer, Dordrecht.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

Herbert, E.A., Dowsland, K.A., 1996. A family of genetic algorithms for the pallet loading problem. Annals of Operations Research 63, 415–436.

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

Lodi, A., Martello, S., Vigo, D., 1999. Heuristic and Metaheuristic Approaches for a Class of Two-Dimensional Bin Packing Problems. INFORMS Journal of Computing 11 (4), 345–357.

Lodi, A., Martello, S., Monaci, M., 2002. Two-Dimensional Packing Problems: A survey. European Journal of Operational Research 141, 241–252.

Martello, S., Toth, P., 1990. Knapsack Problems. John Wiley, Chichester.

Mori, Y., Tanaka, M., 2002. A hybrid genetic algorithm for timetabling of conference programs. In: Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), pp. 421–440, ISBN 90-806096-1-7.

Nagar, A., Heragu, S.S., Haddock, J., 1995. A meta-heuristic algorithm for a bi-criteria scheduling problem. Annals of Operations Research 63, 397–414.

Ozcan, E., Alkan, A., 2002. Timetabling using a steady state genetic algorithm. In: Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), pp. 104–106, ISBN 90-806096-1-7.

Pardalos, P., Resende, M., 2002. Handbook of Applied Optimization. Oxford University Press.

Prosser, P., 1988. A hybrid genetic algorithm for pallet loading. In: Proceedings of the Eight European Conference on Artificial Intelligence. Pitman, London, pp. 159–164.

Tamura, H., Hirahara, A., Hatono, I., Umano, M., 1994. An approximate solution method for combinatorial optimisation. Transactions of the Society of Instrument and Control Engineers 130, 329–336.