How to Pack Trapezoids: Exact and Evolutionary Algorithms

Rhyd Lewis and Penny Holborn

***Abstract*—The purposes of this paper are twofold. In the first, we describe an exact polynomial-time algorithm for the pair sequencing problem and show how this method can be used to pack fixed-height trapezoids into a single bin such that inter-item wastage is minimised. We then go on to examine how this algorithm can be combined with bespoke evolutionary and local search methods for tackling the multiple-bin version of this problem—one that is closely related to one-dimensional bin packing. In the course of doing this, a number of ideas surrounding recombination, diversity, and genetic repair are also introduced and analysed.**

***Index Terms*—Trapezoid Packing, Bin Packing, Pair Sequencing Problem, Recombination, Diversity.**

I. Introduction

The *Pair Sequencing Problem* (PSP) is defined as follows:

Definition 1. *Let $\mathcal{P}$ be a multiset of unordered pairs of nonnegative integers $\mathcal{P} = \{\{x_1, y_1\}, \{x_2, y_2\}, \ldots, \{x_n, y_n\}\}$, and let $\mathcal{X}$ be an ordering of the elements of $\mathcal{P}$ in which each element is also expressed as an ordered pair. The PSP involves identifying the solution $\mathcal{X}$ which minimises the objective function*

$$f(\mathcal{X}) = \left(\sum_{i=1}^{n-1} D\Big(\mathbf{rhs}(i), \mathbf{lhs}(i+1)\Big)\right) + D\Big(\mathbf{rhs}(n), \mathbf{lhs}(1)\Big) \quad (1)$$

where $\mathbf{lhs}(i)$ and $\mathbf{rhs}(i)$ denote the values on the left- and right-hand sides of the ith ordered pair in $\mathcal{X}$, and where $D(x, y) = |x - y|$ denotes the difference *between two values $x, y \in \mathbb{N}_0$.*

The PSP can be used in the game of dominoes to determine whether a set of tiles can be laid out legally in a single (non-branching) line of play. This is achieved by using each $\{x_i, y_i\} \in \mathcal{P}$ to represent a tile with "end" values x_i and y_i, with a legal line of play then corresponding to a solution $\mathcal{X}$ in which at most one of the terms in the objective function has a non-zero value (see Fig. 1). Indeed, if the cost of $\mathcal{X}$ is zero then the ends of the two terminal dominoes can also be joined to form a circuit. In a similar fashion, the PSP can also be used to determine whether a set of n matrices with dimensions x_i, y_i $(i = 1, \ldots, n)$ can be ordered and transposed so that they might be properly multiplied together (though, of course, different solutions may bring about different answers to the resultant calculation).

R. Lewis is with the School of Mathematics, Cardiff University, Wales, e-mail: LewisR9@cf.ac.uk.

P. Holborn is with the School of Computing and Mathematics, University of South Wales, Wales, e-mail: Penny.Holborn@southwales.ac.uk.

Fig. 1. A legal line of play in dominoes. Here, $\mathcal{P} = \{\{1,4\}, \{1,6\}, \{2,4\}, \{2,5\}, \{3,5\}\}$ and the solution is written $\mathcal{X} = \langle(6,1), (1,4), (4,2), (2,5), (5,3)\rangle$, giving $f(\mathcal{X}) = 3$.

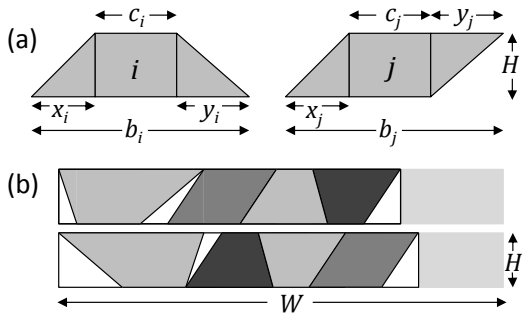

Fig. 2. (a) Examples of form-1 (left) and form-2 (right) trapezoids; and (b) two example packings of the same four trapezoids into an $H \times W$ bin. Inter-item wastage is shown in white.

Our main motivation for studying the PSP, however, comes from a variant of the one-dimensional bin packing problem that was originally considered by Lewis et al. in 2011 [1]. In their work the items to be packed have a fixed height, vary in width, and are trapezoidal in shape. The problem is of particular interest in the construction industry, where we are interested in cutting trapezoidal-shaped roof trusses from fixed-length rectangular stocks; however, it also has other applications, such as when laying decked flooring (see Section V).

Consider a set U of trapezoidal items of a fixed height H. Each item $i \in U$ is defined as having a "base width" b_i, and two "projections" x_i and y_i that determine the angles of its lateral sides. An item's "central width" is simply $c_i = b_i - (x_i + y_i)$. Each trapezoid can also be one of two forms: form-1, where projections occur on the same side of the shape, or form-2, where they occur on alternate sides. In both cases the area of an item i is simply $A(i) = \frac{1}{2}H(b_i + c_i)$ (see Fig. 2(a)).

Definition 2. *Given a set U of trapezoidal items of height H and base widths $b_i \leq W, \forall i \in U$, the trapezoid packing problem (TPP) involves packing the items of U into a minimal number of $H \times W$ bins such that no bin is over-filled.*

Note that the classical one-dimensional bin packing problem is a special case of the TPP in which $x_i = y_i = 0, \forall i \in U$; hence the TPP is NP-hard.

In addition to the bin packing task of deciding which items should be assigned to which bins, the TPP involves deciding *how* items should be packed into each bin such that wastage between successive items is minimised. This task can be formally defined as follows.

Definition 3. *Let $S \subseteq U$ be a set of trapezoidal items whose total area is less than or equal to a bin's area (i.e., $A(S) = \sum_{i \in S} A(i) \leq HW$). The* trapezoid packing subproblem (TPSP)*, involves determining whether an arrangement of the items in S exists such that the "inter-item wastage" is less than or equal to $HW - A(S)$.*

Here, inter-item wastage is defined as the total area of all triangular spaces between each pair of adjacent items, plus the left- and right-most triangles of wastage, as illustrated in Fig. 2(b). If the inter-item wastage is indeed less than or equal to $HW - A(S)$, then it is obvious that an arrangement of the items exists that allows them to be packed into a single bin.

In [1], the TPSP was noted as being a special type of travelling salesman problem and was conjectured to be NP-complete. These observations were then used to justify a greedy approximation algorithm for the subproblem (discussed further in Section III-B). An exact IP-based model was also used when their greedy algorithm was too inaccurate, though this often turned out to be restrictively slow in their experiments.

In this present work we show that the TPSP can be expressed as a type of PSP. In Section II, we then show that the PSP, and therefore the TPSP itself, can in fact be exactly solved using a polynomially bounded algorithm, therefore disproving Lewis et al.'s conjecture [1]. After presenting this algorithm, in Sections III and IV we then go on to show how this exact method can be combined with state of the art packing heuristics to produce high-quality results for the TPP. Section V then concludes the paper and makes some suggestions for further work.

A. Expressing the TPSP as a PSP

Let us start by making the following observations about the TPSP. First, since we are only attempting to minimise inter-item wastage, the values b_i and c_i can be ignored as they have no bearing on the calculation. Second, although each trapezoid in a sequence can be aligned according to four orientations (by flipping on none, either, or both of its horizontal and vertical axes), only two of these orientations need to be considered due to the following theorem.

Theorem 1 ([1])**.** *When minimising wastage between successive trapezoids in a defined sequence, we only need to decide whether each trapezoid should be flipped on its vertical axis.*

Proof. Suppose a set S of trapezoids have been placed in a particular sequence from left to right, together with a specification, for each trapezoid, of which projection should be on the left. If the orientations are such that the inter-item wastage for this particular arrangement is minimised, then the adjacent projections will be aligned so that they "nest". That is, "$\diagup$" angles will be adjacent to other "$\diagup$" angles and "$\diagdown$"angles will be adjacent to other "$\diagdown$" angles (as is the case in Fig. 2(b)). Now suppose the contrary, and that two adjacent trapezoids in this arrangement do not nest. If we now take all trapezoids to the right of this join and flip them on their horizontal axis, this join will be nested, decreasing this wastage, and leaving the remaining joins in the sequence unchanged. Hence, the original orientation of the items could not have given the minimal wastage. □

The task of arranging a set S of trapezoids into a single bin can now be seen as involving two things: (a) determining their ordering from left to right, and (b) deciding for each trapezoid $i \in S$ whether projection x_i or y_i should occur on the left. Joins between adjacent trapezoids can then be easily nested due to Theorem 1. Note that this allows us to disregard the differences between form-1 and form-2 trapezoids. It also means that the area of wastage between any two projections x_i and x_j can be calculated as $\frac{1}{2}H(|x_i - x_j|)$. This can be further simplified to $|x_i - x_j|$ by assuming $H = 2$, which has no effect on the problem or its solutions.

It is now clear that the task of optimally arranging a set S of trapezoids into a $H \times W$ bin can be expressed as an instance of the PSP using $\mathcal{P} = \{\{x_i, y_i\} : i \in S\} \cup \{\{0, 0\}\}$. Here, the additional element $\{0, 0\}$ is used for calculating the left- and right-hand triangles of wastage and can be viewed as a trapezoid i for which $x_i = b_i = y_i = 0$. Our task is to now identify a PSP solution $\mathcal{X}$ whose cost $f(\mathcal{X}) \leq HW - A(S)$.

II. Solving the PSP

In this section we give a polynomially bounded exact algorithm for the PSP. In proving the correctness of this algorithm it is useful to consider the problem from a graph-theoretic point-of-view.

Definition 4. *Let $\mathcal{P}$ be an instance of the PSP, and let $G = (V, E)$ be an undirected multigraph defined by an edge multiset $E = \mathcal{P}$, giving $|E| = n$. The vertex set V is defined using one vertex for each of the different values occurring in $\mathcal{P}$. That is, $V = \bigcup_{i=1}^{n} \{x_i, y_i\}$. For convenience, let the subscript j of a vertex v_j correspond to its numerical value in $\mathcal{P}$; hence the degree of v_j, written $\deg(v_j)$, corresponds to the number of occurrences of the value j in $\mathcal{P}$.*

Considering a graph G constructed in this manner, the task of forming a solution to the PSP might be viewed as a special type of undirected rural postman problem (RPP). In the RPP we are given an arbitrary edge-weighted graph for which some edges are marked as compulsory. The task is then to form a cycle that traverses all compulsory edges at least once. In cases where all edges are marked as compulsory, the RPP becomes equivalent to the well-known Chinese postman problem, which is solvable in polynomial time [2]; however, the RPP is known to be NP-hard in general [3].

For the PSP, all edges in G are compulsory. Like the RPP, we are interested in forming a cycle containing all compulsory edges, though these must be traversed exactly once. Note that

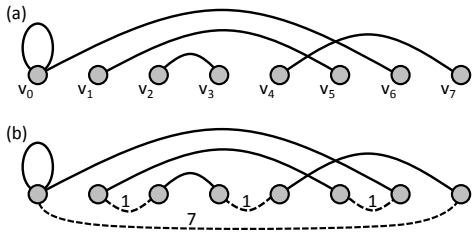

Fig. 3. (a) The graph G formed by the example problem $\mathcal{P} = \{\{0,0\}, \{0,6\}, \{1,5\}, \{2,3\}, \{4,7\}\}$; and (b) an example (sub-optimal) solution $\mathcal{X} = \langle(0,0), (0,6), (5,1), (2,3), (4,7)\rangle$, with $f(\mathcal{X}) = 10$. Solid edges corresponds to the trapezoidal items and are therefore compulsory. Dotted edges incur a cost (indicated).

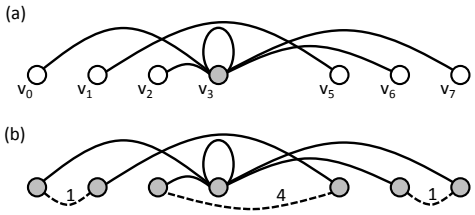

Fig. 4. (a) Graph G formed by the problem $\mathcal{P} = \{\{0,3\}, \{1,5\}, \{2,3\}, \{3,3\}, \{3,6\},\{3,7\}\}$. Odd-degree vertices are shown in white. (b) shows the resultant Eulerian graph when the matching M^* (dotted lines) has been added.

this may require additional edges to be added to G, as Fig. 3 demonstrates. These additional edges will each incur a cost, defined as $w(v_i, v_j) = D(i,j) = |i-j|$. The graph G can therefore be seen as a special type of Euclidean graph in which all vertices lie on a straight line. In particular, given vertices v_i, v_j, v_k with $i < j < k$, this implies $w(v_i, v_k) = w(v_i, v_j) + w(v_j, v_k)$. Solving the PSP using G can therefore be viewed as the task of identifying the set of additional edges whose total cost is minimal, but which allows all edges in G to be traversed exactly once.

We now recall the following definition from classical graph theory:

Definition 5. *A graph is* Eulerian *if and only if it has no vertices of odd degree (that is, all vertices are of even degree).*

An *Eulerian cycle* (or Eulerian tour) of a graph is defined as a cycle that visits every edge exactly once and that starts and ends at the same vertex. Eulerian cycles were introduced by Leonard Euler in the mid-seventeenth century in his solution to the famous Seven Bridges of Königsberg problem [4]. In this work, the following theorem was also stated, a proof of which was later published by Hierholzer and Wiener [5].

Theorem 2 ([4], [5])**.** *A graph contains an Eulerian cycle if and only if it is both connected and Eulerian.*

Theorem 2 now allows us to state the following for the PSP.

Theorem 3. *There exists a zero cost solution $\mathcal{X}$ to an instance $\mathcal{P}$ of the PSP if and only if its corresponding graph $G = (V, E)$ features an Eulerian cycle.*

Proof. Let $\mathcal{C} = \langle(v_{x_1}, v_{x_2}), (v_{x_2}, v_{x_3}), (v_{x_3}, v_{x_4}), \ldots, (v_{x_n}, v_{x_1})\rangle$ be an Eularian cycle in G. In each vertex v_{x_i} encountered along this cycle, we "enter" v_{x_i} via the edge $(v_{x_{i-1}}, v_{x_i})$, and "exit" via the edge $(v_{x_i}, v_{x_{i+1}})$. Since $D(x_i, x_i) = 0$, the corresponding PSP solution $\mathcal{X} = \langle(x_1, x_2), (x_2, x_3), (x_3, x_4), \ldots, (x_n, x_1)\rangle$ has a cost of zero.

Alternatively, let $\mathcal{X}$ be the zero-cost PSP solution $\mathcal{X} = \langle(x_1, x_2), (x_2, x_3), (x_3, x_4), \ldots, (x_n, x_1)\rangle$. Because $\mathcal{P} = E$ and the elements of $\mathcal{X}$ have a one to one correspondence to elements in $\mathcal{P}$, then $\mathcal{C} = \langle(v_{x_1}, v_{x_2}), (v_{x_2}, v_{x_3}), (v_{x_3}, v_{x_4}), \ldots, (v_{x_n}, v_{x_1})\rangle$ defines an Eulerian cycle. □

Eulerian cycles can be constructed from connected Eulerian graphs using Hierholzer's algorithm, which is of complexity $O(n)$ (see [5], [6]). For the PSP an optional step is also available in which we might reduce n by removing certain edges from G before identifying the Eulerian cycles. Specifically,

- If there exist two edges $\{v_x, v_x\}$ and $\{v_x, v_y\}$, these can be replaced by the single edge $\{v_x, v_y\}$. Note that this condition also allows the possibility that $x = y$, in which case multiple loops of the form $\{v_x, v_x\}$ can be replaced by a single loop $\{v_x, v_x\}$.
- If there exist three edges $\{v_x, v_y\}$, $\{v_x, v_y\}$, and $\{v_x, v_y\}$ (where $x \neq y$), these can be replaced by the single edge $\{v_x, v_y\}$.

These opportunities arise from the fact that the above edge combinations can always be adjacent in any Eulerian cycle. For example, in the second bullet above we can always form an Eulerian cycle containing $\langle\ldots(v_x, v_y), (v_y, v_x), (v_x, v_y)\ldots\rangle$; hence, in a graph in which two of these edges have been removed, an Eulerian cycle containing just one occurrence of (v_x, v_y) obviously still exists.

In instances of the PSP for which the corresponding graph G does not feature an Eulerian cycle, it is necessary to add additional edges, each of which will attract an additional cost. This leads to an optimal solution $\mathcal{X}$ for which $f(\mathcal{X}) > 0$. Let $G' = (V', E')$ be a complete edge-weighted subgraph comprising only the odd-degree vertices of G and edge weights $w(v_i, v_j) = D(i,j)$ for all $v_i, v_j \in V'$. According to the handshaking theorem, $|V'|$ must be even [4]. In addition, because the vertices of V' lie on a straight line, a minimum-weight perfect matching M^* can be achieved by considering the vertices from left to right and simply taking the edge that joins each successive pair (as is the case in Fig. 4). The appropriate FIND-MATCHING procedure for this task is shown in Fig. 5. Note that a more expensive weighted matching algorithm for general graphs might also be applied to achieve this task; however, this is unnecessary due to the special structure of G' noted.

Theorem 4. *Let $G = (V, E = \mathcal{P})$ be non-Eulerian. Now set $E \leftarrow E \cup M^*$, where M^* is found by the* FIND-MATCHING *procedure. If G is now connected, an Eulerian cycle of G defines a minimum cost solution to $\mathcal{P}$.*

Proof. It is obvious that G is now Eulerian since all of the originally odd-degree vertices in G have had their degrees increased by one. Because G is connected, it therefore contains an Eulerian cycle according to Theorem 2. If an Eulerian cycle of G does not define a minimum cost solution to the PSP, this

```
FIND-MATCHING (G = (V, E = P))
(1)  Let V = {v_{x_1}, v_{x_2}, ..., v_{x_l}} where x_1 < x_2 < ... < x_l
(2)  M* ← ∅, i ← 1
(3)  while (i ≤ l) do
(4)      if deg(v_{x_i}) is odd then
(5)          for j ← i + 1 to l do
(6)              if deg(v_{x_j}) is odd then
(7)                  M* ← M* ∪ {{v_{x_i}, v_{x_j}}}
(8)                  break
(9)          i ← j + 1
(10)     else i ← i + 1
```

Fig. 5. Algorithm for achieving a minimum-weight matching M^* on the odd-degree vertices of G.

implies the existence of a weighted matching M whose edge weight sum is less than M^*. However, since all vertices of G lie on a straight line and have Euclidean distances, such a matching can obviously not exist. □

As a final step, we now need to consider the situation where the Eulerian graph G comprises more than one component. If this is the case, these Eulerian components will need to be joined by adding further edges to the graph. To see how this can be done, it is instructive to view each component as a *sub-solution* to the original PSP problem $\mathcal{P}$. Note that any sub-solution (and indeed full solution) to the PSP remains unchanged under cyclic shifts and inversions. For example, the solution $\langle(1,2),\ (3,4),\ (5,6),\ (7,8)\rangle$ is equivalent to the solution $\langle(7,8),\ (1,2),\ (3,4),\ (5,6)\rangle$ (due to a right cyclic shift), and the solution $\langle(8,7),(6,5),(4,3),(2,1)\rangle$ (due to an inversion). This means that a particular sub-solution X_2 can be inserted into another sub-solution X_1 in $2 \times |X_1| \times |X_2|$ different ways. We define such an operation as a *splice*.

In more detail, let the sub-solutions $X_1 = \langle \ldots (x_i, x_{i+1}),$ $(x_{i+2}, x_{i+3}) \ldots\rangle$ and $X_2 = \langle (y_1, y_2), \ldots,\ (y_{l-1}, y_l)\rangle$. Splicing these sub-solutions by inserting X_2 between elements (x_i, x_{i+1}) and (x_{i+2}, x_{i+3}) in X_1 results in a new sub-solution with an additional cost of

$$D(x_{i+1}, y_1) + D(x_{i+2}, y_l) - D(x_{i+1}, x_{i+2}) - D(y_1, y_l). \quad (2)$$

Alternatively, inserting an inverted X_2 in the same manner gives a sub-solution with an additional cost

$$D(x_{i+1}, y_l) + D(x_{i+2}, y_1) - D(x_{i+1}, x_{i+2}) - D(y_1, y_l). \quad (3)$$

In each of these cases, there are $|X_1|$ possible insertion points for X_2 in X_1, and $|X_2|$ possible versions of X_2 due to cyclic shifts. This gives the $2\times|X_1|\times|X_2|$ possible options as stated.

Definition 6. *Let X_i and X_j be two sub-solutions. A* minimum cost splice *is the operation of splicing X_i and X_j such that the minimum additional cost, denoted by $\rho(X_i, X_j)$, is incurred.*

A minimum cost splice between two sub-solutions X_i and X_j can be calculated by simply checking all $2 \times |X_i| \times |X_j|$ possible options and taking the smallest value. The act of performing this minimum cost splice is denoted by $X_i \leftarrow$ SPLICE(X_i, X_j). That is, X_j is simply copied into X_i in the appropriate way at the correct position. As an example, consider two sub-solutions $X_1 = \langle(0,2),\ (2,4),\ (4,3),$ $(3,2)\rangle$ and $X_2 = \langle(6,7),\ (7,5),(5,6)\rangle$. The operation $X_1 \leftarrow$ SPLICE(X_1, X_2) garners an additional cost of $\rho(X_1, X_2) = 2$, which involves applying one right cyclic shift to X_2 and then inserting it between the second and third elements of X_1 to give $X_1 = \langle(0,2),\ (2,4),\ (\mathbf{5},\mathbf{6}),\ (\mathbf{6},\mathbf{7}),\ (\mathbf{7},\mathbf{5}),\ (4,3),\ (3,2)\rangle$.

```
MERGE-SUB-SOLUTIONS (X = {X_1, X_2, ...})
(1)  while |X| > 1 do
(2)      Determine X_i, X_j ∈ X with minimal ρ(X_i, X_j), i < j
(3)      X_i ← SPLICE(X_i, X_j)
(4)      X ← X − {X_j}
(5)  X ← X_1
```

Fig. 6. Algorithm for optimally merging all subsolutions into a single, complete solution.

An algorithm for merging all sub-solutions into a single, full solution to the PSP is shown in Fig. 6. Given a set of sub-solutions $\mathcal{X} = \{X_1, X_2, \ldots\}$, in each iteration the pair of sub-solutions with the minimum cost splice overall is identified and these are spliced together appropriately. The final full solution results when only one sub-solution remains, as shown in Line (5) of Fig. 6. Note that this algorithm is analogous to Kruskal's algorithm for identifying a minimum spanning tree [7]. This operates by taking an arbitrary edge-weighted graph and, at each step, merging a pair of components, until only one component (representing a minimum spanning tree) remains. More specifically, at the outset Kruskal's algorithm considers each vertex of the graph as a component, and then selects the lowest cost edge between any two components. The components at the end points of this edge are then merged into a single component, and the process then repeats.

From the perspective of using the multigraph G to represent a PSP, because all vertices occur in a single row, each pair of Eulerian components $X_i, X_j \in G$ can be seen as being linked by an edge with cost $\rho(X_i, X_j)$. The resultant spanning tree then describes the way in which these components are to be merged together. In effect, however, each application of the operation $X_i \leftarrow$ SPLICE(X_i, X_j) is actually adding two edges between X_i and X_j (whose total weight equals $\rho(X_i, X_j)$) to ensure that the resultant component is still Eulerian.

The final overall procedure for solving the PSP, which we call the EULER-SPLICE algorithm, is as follows:

1) Given $\mathcal{P}$, form the graph $G = (V, E)$ according to Definition 4. If G is connected and Eulerian, return an Eulerian cycle and end.
2) If $G = (V, E)$ is not Eulerian, determine the matching M^* by executing FIND-MATCHING(G) and set $E \leftarrow E \cup M^*$. If G now comprises one component, return an Eulerian cycle of G and end.
3) Execute MERGE-SUB-SOLUTIONS$(\mathcal{X})$, where the input $\mathcal{X} = \{X_1, X_2, \ldots\}$ is the set of all Eulerian components of G. Return $\mathcal{X}$ as the optimal solution to $\mathcal{P}$.

An example application of this method is provided in Fig. 7. Given $|\mathcal{P}| = n$, observe that the FIND-MATCHING procedure is of linear complexity $O(n)$, as is the act of determining the resultant Eulerian cycles via Hierholzer's algorithm [5], [6]. MERGE-SUB-SOLUTIONS can involve up to n iterations of the outer while-loop, while Line (2) of the procedure is of

complexity $O(n^2)$ due to the number of calculations that need to be carried out in calculating the minimum $\rho(X_i, X_j)$ across all pairs of components $X_i, X_j \in \mathcal{X}$. Splicing operations, however, are of complexity $O(1)$ if linked lists (or similar) are used. EULER-SPLICE therefore features an overall worse case complexity $O(n^3)$

III. INITIAL RESULTS WITH THE TPP

From this point of the paper onwards we now focus on algorithms for the trapezoid packing problem (TPP), making particular use of the EULER-SPLICE method for solving the TPSP (i.e., for determining the best item packings in individual bins). Recall from Definition 2 that the TPP involves packing a set of fixed height trapezoidal items U into a set of bins such that (a) no bin is overfilled (i.e., the packing in each bin is *feasible*), and (b) the number of bins used is minimised.

In general, our approach to this multi-bin problem will use ideas stemming from the one-dimensional bin packing problem, though suitable modifications are also needed to cope with the trapezoidal nature of the items. Here, a feasible candidate solution is denoted by the set $\mathcal{S} = \{S_1, \ldots, S_k\}$ such that (a) $\bigcup S_i = U$; (b) $S_i \cap S_j = \emptyset$ (for all $i \neq j$); and (c) for all $S_i \in \mathcal{S}$, the items in S_i can be feasibly packed into a single bin. Obviously, k is the variable that we seek to reduce.

The one-dimensional bin packing problem (BPP) has been the target of much research in the past fifty years, with a number of different approximation algorithms and heuristics being proposed [8]–[12]. Perhaps the simplest approximation algorithm is the first-fit (FF) algorithm, which operates by taking each item in U one by one in an arbitrary order and assigning it to the lowest indexed bin into which it can be feasibly packed, opening new bins when necessary. FF has an asymptotic worst case ratio of $\frac{17}{10}$ [13]. A simple improvement on this is the first-fit *descending* (FFD) heuristic, which sorts the items into descending order of size (area) before applying FF as before, featuring an asymptotic worst case ratio of $\frac{11}{9}$ [14]. Better still is the ratio $\frac{71}{60}$ achieved by the more time-consuming algorithm of Garey and Johnson [15].

The FF algorithm can also be used as a basis for the first-fit *grouping* (FFG) heuristic. This involves taking an existing feasible solution $\mathcal{S} = \{S_1, \ldots, S_k\}$ and forming an ordering of the items such that items within the same bin S_i ($\forall S_i \in \mathcal{S}$) are adjacent to one another.[1] It is known that if such an ordering is used with FF, the resultant solution $\mathcal{S}'$ will feature an equal number or fewer bins than the original solution $\mathcal{S}$ [9]. For these reasons, it is also known that there exists at least one item ordering with which FF produces an optimal solution, since such an ordering can be generated using an optimal solution itself.

Despite its obvious overlap with the BPP, note that the above worst case ratios do not hold for the TPP due to the (often necessary) inter-item wastage that can occur within solutions. For similar reasons, the well-known *theoretical minimum* number of bins in a BPP solution, $\text{TMin} = \lceil (\sum_{i \in U} A(i))/HW \rceil$, will also be less accurate, particularly for problems whose solutions involve only a small number of items per bin. Nevertheless, in this section we choose to look at how the FFD bin packing heuristic, suitably modified for trapezoidal items, performs across a large set of benchmark problem instances for the TPP.

A. Problem Instances

The TPP problem set used here was created in [1] and comprises 1,300 problem instances intended to model real-world truss cutting problems encountered in the construction industry (the original inspiration for the problem). The number of items $|U|$ ranges from 100 to 500 and bin size W is set to 4,200 mm, which is a standard industry size. As explained in Section I-A, $H = 2$ in all cases. Item widths b_i are set to between 300 and 3600 mm using values selected uniform randomly within this range, or so that a particular number of "large" items (2,700–3,600 mm) and "small" items (300–1,800 mm) are included. Projection sizes are then determined by setting the angles of the lateral sides to between 30°and 90°, while ensuring that $x_i + y_i \leq b_i$ for all $i \in U$. Within this set, two kinds of problems are also included: artificial ("a") instances, where each item has its own unique dimensions, and the more realistic ("r") instances where sets of identical items are present within each instance.[2]

It is noted in [1] that solutions to these problem instances tend to feature fewer than three items per bin on average (that is, the trapezoidal items tend to be quite long and thin), meaning that the TPSPs encountered during a run are often very small. To gain a wider view of performance, this "original" instance set has therefore been modified to create two further sets in which the central width c_i of each item $i \in U$ is set to half and a quarter of their original values. The main features of these instance sets are given in the first four columns of Table I.

B. Tackling the Trapezoid Packing Sub-Problem (TPSP)

As we have seen, at various points during execution it is necessary to determine whether a subset of items $S \subseteq U$ can be feasibly packed into a particular bin. For one-dimensional bin packing, this involves simply checking whether the total size (area) of the items in S is less than or equal to the bin capacity. For the TPP this criterion also applies, though further computation is also needed to determine whether a solution to the TPSP exists. In their original work, Lewis et al. [1] suggested first using the following simple checks to identify the existence (or otherwise) of such a solution. These were said to be sufficient for solving approximately half of the TPSPs encountered during their experiments.

- If $|S| = 1$ then a feasible packing exists.
- If $A(S) = HW$, then it is necessary for a feasible packing to feature no inter-item wastage; consequently, in S there needs be an even number of occurrences of each projection size, and there should be at least two projections of size zero (to ensure that no wastage occurs

[1] Note that $k!$ such orderings of the bins are available here. Whenever FFG is applied in this paper, one of these $k!$ orderings will be selected at random.

[2] A more detailed description of this generator, together with the instances themselves, is available at [16].

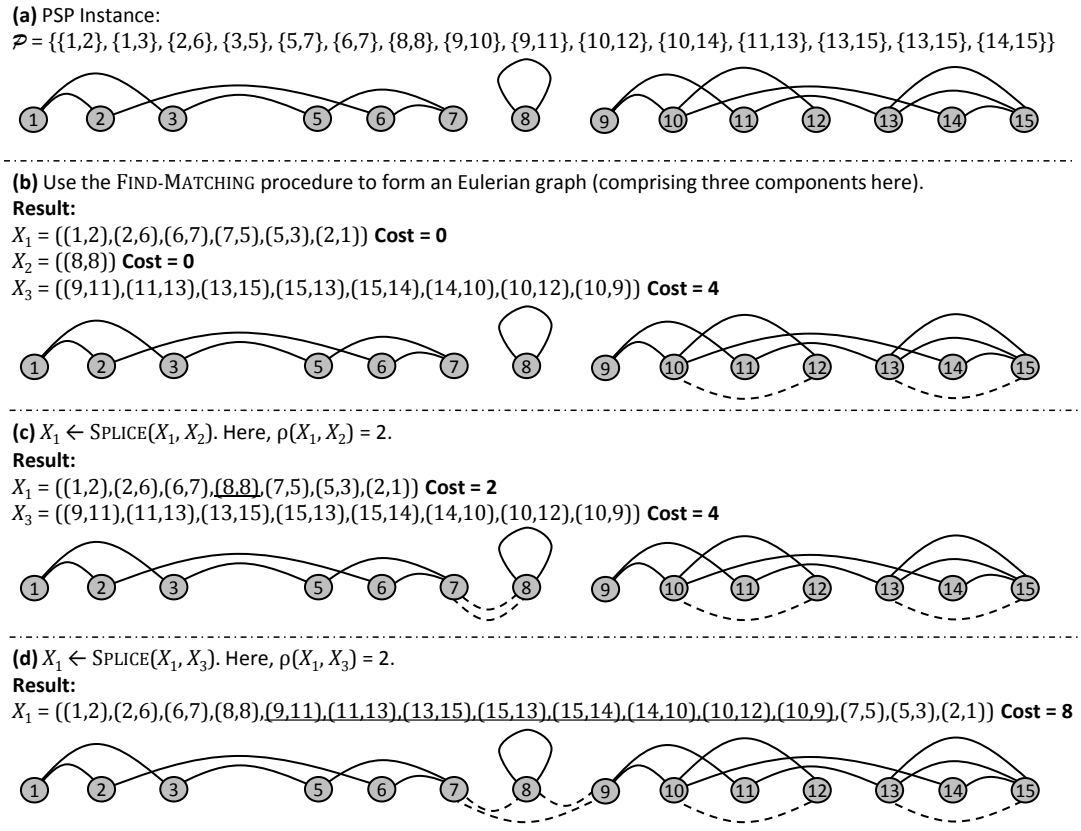

Fig. 7. Example application of the EULER-SPLICE algorithm.

in the left and right extremes of the bin). If this is not the case then no feasible packing exists.

- Let τ_i denote the minimum amount of wastage we can hope to occur around item $i \in S$ in a packing, determined by inspecting the closest projection sizes to x_i and y_i in the set $S - \{i\}$. Now let $\tau_{\max} = \max(\tau_i : i \in S)$. It is obvious that if $\tau_{\max} > HW - A(S)$ then no feasible packing exists.

In cases where these basic checks were inconclusive, Lewis et al. [1] made use of the following GREEDY method to try and determine a feasible packing of the items in S.

1) Make a copy of S called S'. Given S', identify the item $i \in S'$ with the smallest projection. Place this item into the left-most position in the bin with the identified projection on the left hand side, and remove i from S'.
2) Consider the size of the right projection of the right-most item in the bin. Call this value x. Now identify the item $i \in S'$ with a projection size y closest to x (i.e., that minimises $|x - y|$). Add item i to the bin so that the y-sized projection is nested with the projection of size x. Remove i from S' and repeat Step 2 until $S' = \emptyset$.

If, on completion of these steps, the inter-item wastage was seen to be less than or equal to $HW - A(S)$ then a feasible packing was known to exist. If this was not the case (the bin was overflowing) then a further process was invoked by Lewis et al. [1]. This went through each item in the bin from left to right and identified whether rotating the item on its vertical axis reduced the total amount of inter-item wastage. If this was the case, the rotation was performed. At the end of this process, if the inter-item wastage was still larger than $HW - A(S)$, then it was assumed that no feasible packing for S existed.

Of course, since the GREEDY algorithm is only approximate, it is natural here to replace it with the exact EULER-SPLICE method. Our implementation of EULER-SPLICE operates as described in Section II but without the optional step of edge removal. This is because the TPSPs, and therefore individual instances of the PSP, encountered in the benchmark problems were found to be quite small on the whole (usually less than ten items) meaning that the overheads of this stage usually outweighed the expense of simply applying EULER-SPLICE to the original problem. This will not be the case for larger PSP instances, however.

C. FFD Results

Table I compares the results gained by the FFD heuristic, when used in conjunction with either GREEDY or EULER-SPLICE, on the complete set of 3,900 problem instances. All individual trials took less than 15 ms to execute.[3] We see that the FFD variant using EULER-SPLICE produces solutions using fewer bins on average for 26 of the 30 instance subclasses. In 16 of the 26 cases, these differences are statistically

[3]Algorithms presented in this work were written in C++ and executed on a 3.3 GHtz Windows 7 machine with 8 GB RAM. A complete version of the code, together with a complete listing of the results from our trials can be downloaded from www.rhydlewis.eu/resources/trapezoid.zip.

significant, as indicated. Where significance is not observed, note that the sample sizes are quite small due to only 20 problem instances being available.

Table I also demonstrates that solutions using TMin bins tend to be found less frequently when $|U|$ is large and/or when the number of items per bin is small, as with the original instances. This is to be expected since the FFD algorithm will usually be less accurate with larger problem sizes, while small numbers of items per bin tends to cause TMin to underestimate the true optimum. We also see that the variance in the number of bins required is higher for the "r" instances. This is because, with smaller numbers of different item and projection sizes, it is usually more difficult to achieve tight packings of the items.

In summary, the results provide clear evidence that, when packing the trapezoidal items into bins, the use of an exact method to solve the TPSP brings definite advantages over the approximate GREEDY method. An illustration of these improvements is shown in Fig. 8. In this example, observe that the solutions are identical up to bin 12, but in the 13th the EULER-SPLICE method has allowed one further item to be added compared to GREEDY, resulting in one fewer bin overall once all remaining items have been inserted.

IV. IMPROVING RESULTS USING EVOLUTIONARY METHODS

In this section we improve on the results of the previous by employing specialised evolutionary methods, which are known to have produced some of the best-known results for the BPP and related partitioning problems [8], [17]–[20]. Here, we focus particularly on the issue of recombination, with each operator being applied within a common evolutionary framework incorporating a bespoke local search procedure. This procedure is described in the next subsection, with Sections IV-B and IV-C detailing the remaining elements of the evolutionary algorithm (EA). Sections IV-D and IV-E then describe the results of our experiments and explore how further improvements to our methods can be achieved. Note that from this point onwards the EULER-SPLICE algorithm as described in Section III-B is used exclusively for solving the TPSP.

A. Local Search

The local search method employed in all of our algorithms operates on a pair of feasible sub-solutions $\mathcal{S}$ and $\mathcal{S}'$ that, together, make up a full solution. That is, $(\bigcup_{S\in\mathcal{S}} S) \cup (\bigcup_{S'\in\mathcal{S}'} S') = U$, and $(\bigcup_{S\in\mathcal{S}} S) \cap (\bigcup_{S'\in\mathcal{S}'} S') = \emptyset$.

The following steps are applied. For each $S \in \mathcal{S}$ and $S' \in \mathcal{S}'$ (considered in a random order):

1) If there exist pairs of items $i, j \in S$ and $i', j' \in S'$ such that $A(i) + A(j) < A(i') + A(j')$ and $((S \cup \{i', j'\}) - \{i, j\}) \in \mathcal{F}$ and $((S' \cup \{i, j\}) - \{i', j'\}) \in \mathcal{F}$, then move items i, j from S to S' and items i', j' from S' to S.
2) If there exists a pair of items $i, j \in S$ and an item $i' \in S'$ such that $A(i)+A(j) \leq A(i')$ and $((S\cup\{i'\})-\{i, j\}) \in \mathcal{F}$ and $((S' \cup \{i, j\}) - \{i'\}) \in \mathcal{F}$, then move items i, j from S to S' and item i' from S' to S.
3) If there exist items $i \in S$ and $i' \in S'$ such that $A(i) < A(i')$ and $((S\cup\{i'\})-\{i\}) \in \mathcal{F}$ and $((S'\cup\{i\})-\{i'\}) \in \mathcal{F}$, then move item i from S to S' and item i' from S' to S.
4) If there exists an item $i' \in S'$ such that $(S \cup \{i'\}) \in \mathcal{F}$ and $(S' - \{i'\}) \in \mathcal{F}$, then move item i' from S' to S.

Here, the notation $S \in \mathcal{F}$ signifies that all items in S can be feasibly packed into a single bin.[4] This is determined as described in Section III-B.

The above steps are a modified version of a procedure previously used for the BPP based on the concept of bin *dominance*, defined by Martello and Toth [8]–[11]. The idea is that items are interchanged between bins such that wastage within bins in $\mathcal{S}$ decreases, while the number of items in each bin in $\mathcal{S}$ remains the same or also decreases. If this is achieved, then the bins in $\mathcal{S}$ can be said to have improved, while the smaller, easier to pack, items will have been moved into bins in $\mathcal{S}'$. The procedure operates by first attempting to swap a pair of items from a bin in $\mathcal{S}$ with a pair of items from a bin in $\mathcal{S}'$ (Step 1). Similarly, Steps 2 and 3 involve swapping a pair of items with a single item, and then a single item with a single item. In Step 4, attempts are then made to try and transfer a single item from a bin in $\mathcal{S}'$ into a bin in $\mathcal{S}$. If this is possible, and $|S'| = 1$, this reduces the number of bins in $\mathcal{S}'$ by one.

In our application, whenever one of the above four steps are satisfied, the associated items are moved between the bins and the procedure moves immediately back to Step 1. This process continues until none of the four steps are satisfied (that is, an entire parse of the procedure is performed with no changes being made to any bin in $\mathcal{S}$ or $\mathcal{S}'$). Following this, a complete solution is then reconstructed by executing FFG with $\mathcal{S}'$ to get a new partial solution $\mathcal{S}''$ (giving $|\mathcal{S}''| \leq |\mathcal{S}'|$) and then simply setting $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}''$. Note that this entire procedure cannot increase the number of bins being used in a solution, but it does have the potential to decrease it.

B. Recombination

In general, the ideal aim of an EA's recombination operator is to allow different parts (building blocks) of existing candidate solutions to be effectively combined to make new, good-quality offspring solutions. For the BPP it is usually the bins themselves, together with their associated items, that are considered the relevant building blocks, as discussed, for example, in [8], [21], [22]. However, it is not always possible to copy complete bins of items from multiple parents into an offspring because the items occurring in a single bin in one parent might be spread across many bins in another parent. Consequently, offspring solutions will often feature duplicated or missing items, which will then need to be dealt with by some sort of repair operator. Decisions on *how many* and *which* bins to copy from each parent during recombination also need to be made, which could further influence algorithm performance.

The first recombination operator we investigate originates from the grouping genetic algorithm (GGA) of Falkenauer [8] and operates as follows. Given two feasible parent solutions,

[4]That is, $\mathcal{F}$ denotes the set of all item combinations that can be feasibly packed into a single bin.

TABLE I

INSTANCE SET CHARACTERISTICS AND RESULTS GAINED BY THE TWO FFD VARIANTS. THE LOWEST MEAN VALUES FROM THE BINS COLUMNS ARE MARKED IN BOLD. ASTERISKS INDICATE STATISTICAL SIGNIFICANCE AT ≤ 0.05 (*) AND ≤ 0.01 (**) ACCORDING TO A TWO-TAILED PAIRED T-TEST AND TWO-TAILED MCNEMAR'S TEST FOR THE BINS AND TMIN % COLUMNS RESPECTIVELY.

				FFD with GREEDY		FFD with EULER-SPLICE	
Type, $\|U\|$	# Inst.	# Types[a]	TMin[b]	Bins[c]	TMin %[d]	Bins	TMin %
orig., a, 100	20	100	44.65	46.45 ± 6.3	0.0	**46.40** ± 6.3	0.0
orig., a, 200	20	200	90.70	94.95 ± 5.4	0.0	94.95 ± 5.4	0.0
orig., a, 300	20	300	134.70	139.30 ± 3.9	0.0	139.3 ± 3.9	0.0
orig., a, 400	20	400	177.75	183.15 ± 4.5	0.0	**183.10** ± 4.6	0.0
orig., a, 500	20	500	222.55	228.35 ± 3.0	0.0	228.35 ± 3.0	0.0
orig., r, 100	240	20.19	38.66	42.53 ± 28.0	1.6	*__42.51__ ± 28.0	2.1
orig., r, 200	240	20.31	75.50	82.73 ± 27.8	0.0	**__82.70__ ± 27.8	0.0
orig., r, 300	240	20.05	115.10	126.48 ± 27.0	0.0	**__126.41__ ± 27.1	0.0
orig., r, 400	240	20.65	154.44	169.81 ± 27.2	0.0	**__169.69__ ± 27.3	0.0
orig., r, 500	240	19.60	193.03	212.80 ± 28.6	0.0	**__212.60__ ± 28.6	0.0
half, a, 100	20	100	23.45	24.10 ± 4.6	35.0	24.1 ± 4.6	35.0
half, a, 200	20	200	47.45	48.60 ± 3.0	10.0	*__48.30__ ± 2.9	15.0
half, a, 300	20	300	70.55	71.70 ± 2.4	5.0	**71.60** ± 2.5	10.0
half, a, 400	20	400	92.95	94.65 ± 2.3	0.0	*__94.45__ ± 2.5	0.0
half, a, 500	20	500	116.40	118.25 ± 2.4	0.0	**118.15** ± 2.4	0.0
half, r, 100	240	20.19	20.52	21.30 ± 22.2	29.2	**__21.23__ ± 22.3	**34.6
half, r, 200	240	20.31	39.97	41.43 ± 22.2	6.3	**__41.30__ ± 22.2	8.3.0
half, r, 300	240	20.05	60.80	63.17 ± 22.2	1.6	**__63.00__ ± 22.2	2.5.0
half, r, 400	240	20.65	81.34	84.26 ± 21.6	0.0	**__84.10__ ± 21.6	0.0
half, r, 500	240	19.60	101.75	105.67 ± 23.4	0.0	**__105.41__ ± 23.4	0.0
quar., a, 100	20	100	13.00	13.30 ± 4.3	70.0	**13.20** ± 4.7	80.0
quar., a, 200	20	200	25.80	26.40 ± 4.7	40.0	**26.30** ± 4.5	50.0
quar., a, 300	20	300	38.40	39.15 ± 4.4	25.0	**39.10** ± 4.5	30.0
quar., a, 400	20	400	50.60	51.40 ± 2.6	25.0	*__51.20__ ± 2.7	40.0
quar., a, 500	20	500	63.30	64.30 ± 4.0	5.0	**64.25** ± 3.9	5.0
quar., r, 100	240	20.19	11.48	11.78 ± 20.7	70.0	**__11.74__ ± 20.8	**74.2
quar., r, 200	240	20.31	22.14	22.78 ± 20.1	37.1	**__22.73__ ± 20.1	**42.1
quar., r, 300	240	20.05	33.62	34.54 ± 20.7	22.1	**__34.49__ ± 20.7	24.2
quar., r, 400	240	20.65	44.80	45.95 ± 20.3	15.8	**__45.87__ ± 20.2	17.9
quar., r, 500	240	19.60	56.05	57.58 ± 21.2	5.0	**__57.49__ ± 21.2	6.2

[a]Number of different item types per instance (mean across all instances).
[b]TMin $= \lceil (\sum_{i \in U} A(i))/HW \rceil$ (mean across all instances).
[c]Number of bins per solution (mean across all instances, plus or minus the coefficient of variation (%)).
[d]Percentage of instances where a solution using TMin bins was found.

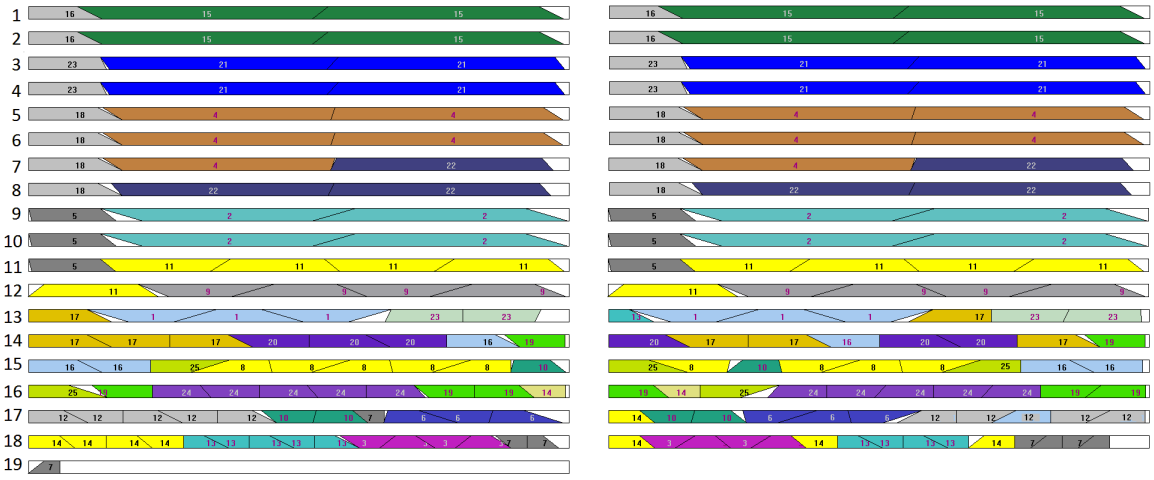

Fig. 8. Example solutions produced for a 100-item "half, r" problem instance using FFD with the GREEDY (left) and EULER-SPLICE (right) algorithms.

$\mathcal{S}_1$ and $\mathcal{S}_2$, the bins of $\mathcal{S}_2$ are first randomly permuted. Two bins in $\mathcal{S}_2$, namely $S_{2,i}$ and $S_{2,j}$ (where $1 \leq i \leq j \leq |\mathcal{S}_2|$), are then randomly selected and all bins between and including these outer bins in $\mathcal{S}_2$ are copied into an offspring solution $\mathcal{S}$, together with all bins from $\mathcal{S}_1$. At this point, $\mathcal{S}$ will contain multiple occurrences of some items, and so the operator goes through $\mathcal{S}$ and deletes any bins containing duplicates that came from parent $\mathcal{S}_1$ (see Fig. 9). This operation results in an offspring solution that comprises only feasible bins, but that could be missing some items. These are then dealt with by a repair procedure, described below.

Our second recombination operator is a modification of the greedy partition crossover (GPX) scheme, which was originally proposed for the graph colouring problem [21]. Unlike the GGA operator, GPX biases the copying of *fuller* bins from parents to offspring. Duplicates are also dealt with differently by only eliminating the offending items themselves, as opposed to entire bins. Specifically, given two feasible parent solutions $\mathcal{S}_1$ and $\mathcal{S}_2$, the fullest bin from either parent is first identified (breaking ties randomly). This bin is then moved into the offspring $\mathcal{S}$ and, to avoid duplicates, the copied items are also removed from the other parent. To form the next

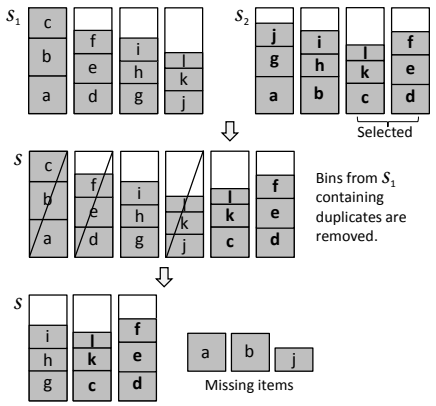

Fig. 9. Example application of the GGA recombination operator, where the third and fourth bins of $\mathcal{S}_2$ have been chosen for insertion into $\mathcal{S}$.

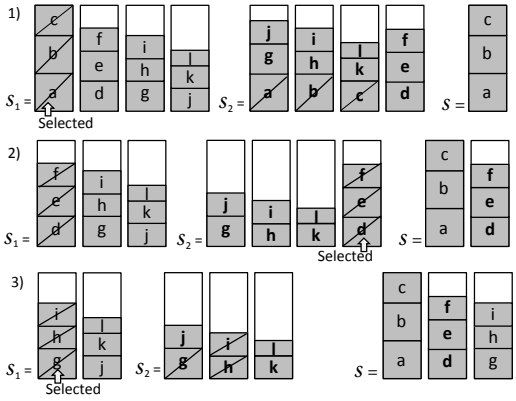

Fig. 10. Example application of the GGA recombination operator. In this case $\min(|\mathcal{S}_1|, |\mathcal{S}_2|) - 1 = 3$ bins are formed in the offspring $\mathcal{S}$, resulting in missing items j, k, and l.

bin, the other (modified) parent is then considered and, again, the fullest bin is moved into $\mathcal{S}$, with these items also being removed from the first parent. This process is continued by alternating between the parents until $\min(|\mathcal{S}_1|, |\mathcal{S}_2|) - 1$ bins have been created in the offspring (see Fig. 10). Missing items are again dealt with by our repair procedure.

Our third recombination operator, which we call GPX', is analogous to that proposed for the BPP by Quiroz-Castellanos et al. [19]. It operates in the same manner as GPX but also uses the GGA's method of eliminating duplicates; hence, once a bin has been copied from a parent to the offspring, rather than simply removing the duplicate *items* from the other parent, any *bins* containing duplicates are deleted. This usually leads to a larger number of missing items compared to GPX.

Upon termination of these recombination operators, an offspring solution $\mathcal{S}$ will comprise only feasible bins, but may be missing some items. To deal with these, our repair operator first uses FFD on the missing items to form a partial solution $\mathcal{S}'$. The local search procedure is then applied using $\mathcal{S}$ and $\mathcal{S}'$ as input to construct a complete solution.

C. EA Framework

Our overall EA works as follows. Given a solution $\mathcal{S} = \{S_1, \ldots, S_k\}$, mutation operates by selecting $0 < l \leq k$ bins at random, removing these from $\mathcal{S}$ and inserting them into a second set of bins $\mathcal{S}'$. The partial solutions $\mathcal{S}$ and $\mathcal{S}'$ are then used as input for the local search procedure, which is used to produce a new, complete solution.

Different values for l have previously been suggested in the BPP literature for similar mutation operators, including "at least three", four, and five (see [8], [11] and [20] respectively). On the other hand, Quiroz-Castellanos et al. [19] have suggested that l should be defined as a stochastic function based on the number of bins k and the proportion of bins not full to capacity, though this is less appropriate for the TPP due to the prevalence of inter-item wastage in solutions. The above four papers also suggest strongly biasing bin selection towards emptier ones; however, we consider this to be unnecessary because (a) it might sometimes be advantageous to break up a full bin, particularly if it is not part of a globally optimal solution; and (b) in some cases, a relatively empty bin might become very high in quality once a small number of other items have been added to it. In our case each application of mutation involves simply selecting a value for l according to the distribution $L \sim 1 + \mathrm{B}(k, 3/k)$, which offers an acceptable compromise.

An initial population for the EA is formed by creating one solution via FFD, and the remainder via FF using random item orderings. Each member of the population is then mutated to try and improve its quality. In each iteration of the EA, two parent solutions are selected at random from the population and recombination is used to create a single offspring. This is then mutated before replacing the least fit of its two parents. The fitness of a parent solution $\mathcal{S}$ is calculated as

$$f(\mathcal{S}) = \frac{\sum_{S \in \mathcal{S}} (A(S)/(HW))^2}{|\mathcal{S}|}, \tag{4}$$

with higher values being deemed fitter. Given two solutions $\mathcal{S}_1$ and $\mathcal{S}_2$, it is known that if $|\mathcal{S}_1| < |\mathcal{S}_2|$ then $f(\mathcal{S}_1) > f(\mathcal{S}_2)$; consequently a global optimum for this function corresponds to a solution with the minimum number of bins [8]. This function is useful because, due to its more fine-grained nature, it allows evolutionary pressure to remain for larger portions of the run compared to simply using the number of bins as a fitness measure. Among solutions using the same number of bins, it also allocates higher fitness values to those whose bins show the highest variance in spare capacity. This encourages some bins in a solution to remain relatively empty, which could be useful in practice if these contiguous areas of wastage were to be used at a later stage, such as when cutting further roof trusses for a different order. Note that exponents larger than two may also be used within this function. This would lead to even higher fitnesses values being awarded to solutions with "extreme" bins (i.e., bins that are either very full, or very empty), although the property regarding global optima stated above no longer holds in such cases.

In extensive preliminary tests, we experimented with a number of different selection and replacement policies, but found that the strategy described above, whose evolutionary

pressure exists solely due to the replacement of the weaker parent, gave the most consistent results. We also used extended runs of up to 1,000 seconds to test a range of different population sizes (5, 10, 25, 50, 100, 200); however, unlike studies with the BPP, which have often used population sizes of 100 or more, we found that the best results, both in terms of final solution quality and the speed of optimisation, were found using the relatively small population size of size 10. This suggests that good solutions are derived from repeated applications of the EA's operators on a small pool of solutions as opposed to a wide sampling of the solution space.

Finally, for comparative purposes, we also implemented the hill-climbing (HC) algorithm of Lewis et al. [1]. This operates on a single solution initially constructed using FFD. In each iteration the mutation operator described above is then employed, followed by the FFG heuristic. Note that, in [1], Lewis et al.'s original version of this algorithm used the GREEDY method for determining item packings as opposed to EULER-SPLICE. Our implementation can therefore be seen as an enhancement on theirs, featuring a more connected solution space.

D. Comparison of Results

Table II compares the quality of the results achieved by our four algorithms across all 3,900 problem instances. In all cases a CPU time limit of 600 seconds was used, which was deemed adequate for providing some notion of excess time in our trials. The number of EA iterations performed within this time limit was found to depend heavily on the amount of time required for each application of the local search operator, which is the most time consuming component of the algorithm. This tends to take longer for problem instances where $|U|$ is high or where there are many items per bin. As a result, the number of EA iterations ranged from approximately 1.2 million to 57,000 on average for the 100-item and 500-item original instances respectively. For the quarter instances, the corresponding figures were just 70,000 and 2,500 iterations.

From Table II it can be seen that the EAs using the GGA and GPX recombination operators produce the best solutions overall; however, their strengths are witnessed in different places. For the original instances, where high quality solutions comprise a relatively small 2.4 items per bin on average, the GGA consistently produces the best results. With the quarter instances on the other hand, where good solutions feature an average closer to 8.8 items per bin, the GPX shows the most favourable behaviour. For the quarter instances, it seems that high quality solutions are achieved by identifying subsets of items that can be packed into a bin with very little wastage—that is, we are interested in determining *individual* bins that are well packed. This mechanism is supplied by the GPX operator, which places a heavy bias on promoting such bins in the population. On the other hand, for the original instances good quality solutions occur more as a result of good *combinations of bins* being identified. Here, the GGA, which features far less bias for promoting well-filled individual bins in the population, facilitates a wider sampling of the solution space, ultimately allowing better results to be achieved.[5] Similar observations to these have previously been made for the graph colouring problem, where an increased edge density of a graph usually increases its chromatic number and, as a result, decreases the number of vertices assigned to each colour [18], [23]. Singh and Gupta [20] have also reported the poor performance of a GPX variant on "triplet" BPP instances (where optimal solutions comprise just three items per bin), presumably for similar reasons.

Table II also shows that the GPX' and HC algorithms are outperformed in all cases by at least one of the two remaining methods. Though GPX' is very similar in form to GPX, its method of deleting entire bins containing duplicates seems to be too destructive and, as a consequence, good building blocks are not being propagated in the population to a sufficient level. HC is also consistently outperformed, suggesting that the use of a population of candidate solutions (with suitable operators) is beneficial compared to using just one. Fig. 11 illustrates the behaviour of the four algorithms over time. The patterns seen in the table appear to be stable and are established early on in the runs. This suggests that shorter or longer run times would not drastically alter the characteristics shown in the table.

1) Population Diversity: It is also instructive to examine the effects of the three recombination operators on population diversity. Diversity measures for partitioning problems have previously been proposed by Mattiussi et al. [24] and Lewis et al. [17]; however, modifications are required with the TPP (and indeed the BPP) due to the interchangeability of items of identical dimensions. In more detail, given an item $i \in U$, let $t(i)$ be an integer denoting its *type*. Two items $i, j \in U$ then have identical dimensions if and only if $t(i) = t(j)$. The items in Fig. 8, for example, are labelled according to type, with bin 1 containing two type-15 items and one type-16 item. The number of item types per instance, as listed in the third column of Table I, is therefore $|\{t(i) : i \in U\}|$. It is obvious that items of the same type can be interchanged within a solution, but that this will have no effect on its underlying structure. To cope with this issue of symmetry, we propose a modification to the diversity measure of [17] as follows.

Definition 7. *Given a solution $\mathcal{S} = \{S_1, \ldots, S_k\}$, let $P_\mathcal{S}$ be a multiset of multisets, where each element of $P_\mathcal{S}$ corresponds to a pair of item types that are assigned to the same bin. That is, $P_\mathcal{S} = \{\{t(i), t(j)\} : i, j \in S \wedge S \in \mathcal{S}\}$.*[6] *The* distance *between two solutions, $\mathcal{S}_1$ and $\mathcal{S}_2$ is then defined according to the Sörensen-Dice measure*

$$d(\mathcal{S}_1, \mathcal{S}_2) = \frac{2|P_{\mathcal{S}_1} \cap P_{\mathcal{S}_2}|}{|P_{\mathcal{S}_1}| + |P_{\mathcal{S}_2}|}, \tag{5}$$

which gives the proportion of elements that occur in both $P_{\mathcal{S}_1}$ and $P_{\mathcal{S}_2}$. (Note that the intersection operator used here refers to the multiset variant.)

[5] Note that statistical significance is only observed with the GGA according to the number of bins used. It is not observed in the TMin % column due to the small sample sizes occurring as a result of TMin's lower accuracy in these cases.

[6] As an example, the left solution from Fig. 8 would result in a multiset containing elements $\{16, 15\}$, $\{16, 15\}$, $\{15, 15\}$, $\{16, 15\}$, $\{16, 15\}$, $\{15, 15\}$, $\{23, 21\}$, $\{23, 21\}$, $\{21, 21\}$, and so on.

TABLE II
RESULTS ACHIEVED AT THE CUT-OFF POINT BY (A) THE GGA, GPX AND GPX' OPERATORS UNDER A COMMON EA FRAMEWORK, AND (B) THE HC ALGORITHM [1]. BOLD TEXT AND ASTERISKS SHOULD BE INTERPRETED AS WITH TABLE I. NOTE THAT RESULTS OF THE "A" AND "R" INSTANCES ARE COMBINED HERE DUE TO THE ALGORITHMS' BEHAVIOUR BEING SIMILAR FOR BOTH CLASSES.

			GGA		GPX		GPX'		HC	
Type, $\|U\|$	# Inst.	TMin[a]	Bins[b]	TMin %[c]	Bins	TMin %	Bins	TMin %	Bins	TMin %
orig., a&r, 100	260	39.12	**3.33**	17.7	3.35	16.5	3.35	16.2	3.36	15.8
orig., a&r, 200	260	76.67	****6.17**	6.5	6.24	6.2	6.23	5.8	6.21	5.4
orig., a&r, 300	260	116.61	****9.62**	1.9	9.80	1.9	9.83	1.2	9.72	1.5
orig., a&r, 400	260	156.23	****12.93**	1.2	13.24	0.0	13.26	0.0	13.10	0.4
orig., a&r, 500	260	195.30	****16.64**	1.2	17.05	0.4	17.09	0.0	16.84	0.0
half., a&r, 100	260	20.75	0.22	78.8	**0.20**	80.4	0.22	79.2	0.26	74.6
half., a&r, 200	260	40.54	0.38	63.8	**0.37**	65.0	0.42	60.4	0.49	53.8
half., a&r, 300	260	61.55	**0.73**	45.4	0.73	46.5	0.82	40.0	0.88	34.6
half., a&r, 400	260	82.23	**0.84**	35.4	0.90	34.6	1.05	25.8	1.05	21.9
half., a&r, 500	260	102.87	***1.16**	23.8	1.25	26.9	1.45	20.0	1.44	13.5
quar., a&r, 100	260	11.60	0.07	93.1	**0.06**	93.8	0.07	93.5	0.08	92.3
quar., a&r, 200	260	22.42	0.12	88.1	***0.08**	**91.5	0.12	88.1	0.15	85.4
quar., a&r, 300	260	33.98	0.17	83.1	***0.14**	**86.2	0.19	81.2	0.24	76.2
quar., a&r, 400	260	45.25	0.21	79.2	***0.17**	**83.1	0.24	76.5	0.27	73.8
quar., a&r, 500	260	56.61	0.32	68.1	****0.27**	**72.7	0.38	63.1	0.43	57.3

[a] TMin $= \lceil (\sum_{i \in U} A(i))/HW \rceil$ (mean across all instances).
[b] Number of bins beyond TMin (mean across all instances).
[c] Percentage of instances where a solution using TMin bins was found.

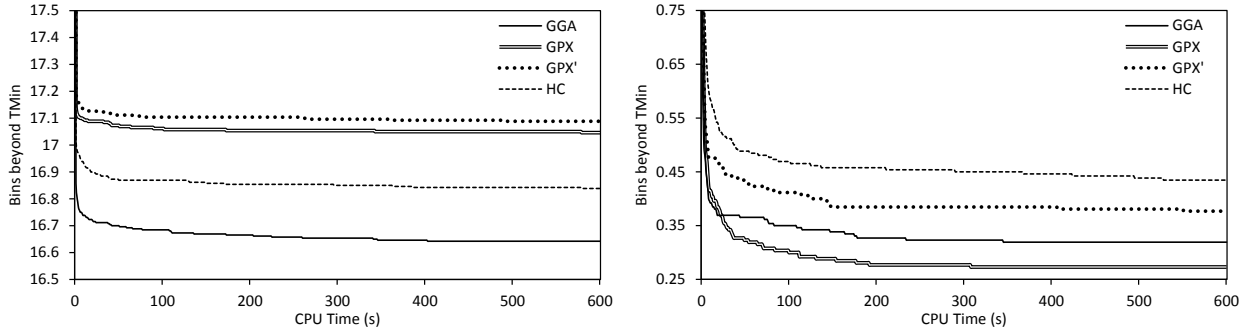

Fig. 11. Run profiles using the $|U| = 500$ original and quarter instances respectively. Each line is the average across all 260 instances at each second.

Definition 8. *Given a population of solutions* $\mathcal{S}_1, \ldots, \mathcal{S}_p$, diversity *is calculated as the mean distance between all pairs of solutions:*

$$D(\mathcal{S}_1, \ldots, \mathcal{S}_p) = \frac{\sum_{\forall \mathcal{S}_i, \mathcal{S}_j : 1 \leq i < j \leq p} d(\mathcal{S}_i, \mathcal{S}_j)}{\binom{p}{2}}. \quad (6)$$

Note that $d(\mathcal{S}_1, \mathcal{S}_2)$, and therefore the diversity measure itself, ranges between 0 and 1, with larger values indicating a higher diversity. An alternative to this method of calculation would be to measure distance according to the number of items that would need to be moved in order to convert one solution into another. However, this scheme heavily depends on the way that bins are labelled in each solution; hence, some sort of bin relabelling scheme of the type reviewed by Coll et al. [25] would also be required.

Figure 12 illustrates how population diversity changes during the first 1,000 iterations of the EA under the three recombination operators. For the original instances it is evident that the two least successful operators, GPX and GPX', maintain higher levels of diversity, suggesting that the algorithms are not suitably "homing in" on the better quality regions of the solution space. Similarly, for the quarter instances it is the GPX operator that shows a steady decrease in diversity while the other operators maintain much higher levels. In this case, the GGA and GPX' operators are not helped by the larger numbers of items per bin, which causes larger numbers of items to become unplaced in each application, therefore making it more difficult to transmit building blocks from one iteration to the next. Such observations have previously been made by Lewis and Paechter [26] who, using a timetabling problem, demonstrate the difficulties experienced by the GGA operator when tackling partitioning problems involving large numbers of items per group.

E. Seeking Further Improvements

In this section we now discuss three possibilities for further improving the results of our EA.

The first of these involves using an updated version of the first-fit (FF) heuristic where, instead of simply ordering the items at random, the "large" items (that is, those of area $A(i) \geq \frac{1}{2}HW$) are placed in a random order in the left-most positions of the item permutation. This heuristic was proposed for the BPP by Quiroz-Castellanos et al. [19], who found that it was able to return better solutions than FF in their tests, albeit using instances featuring high proportions of "large" instances. Note that, with our EA, the incorporation of this heuristic only affects the way in which the initial population is constructed. Additionally, "large" items are only seen to exist in the original instances, and only in small proportions. Consequently, the incorporation of this heuristic was not seen to improve our results in any way.

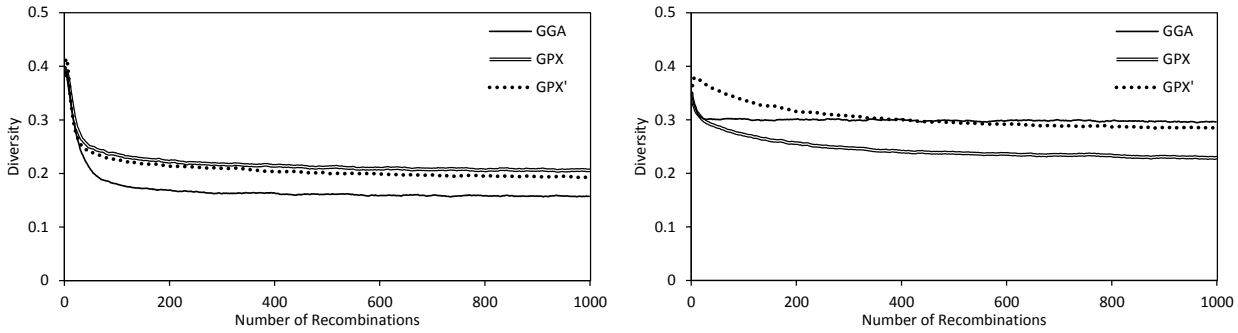

Fig. 12. Population diversities during runs with the $|U| = 500$ original and quarter instances respectively. Each line is the average across all 260 instances.

A more fruitful avenue of research that has yet to be considered within the BPP (and related) literature is to consider the way in which duplicate items are dealt with during recombination. Consider the GGA operator. Given two parent solutions $\mathcal{S}_1 = \{S_{1,1}, \ldots, S_{1,k_1}\}$ and $\mathcal{S}_2 = \{S_{2,1}, \ldots, S_{2,k_2}\}$, recall that an offspring solution is constructed by (a) concatenating all bins from $\mathcal{S}_1$ with $l \leq k_2$ bins from $\mathcal{S}_2$, and then (b) removing duplicated items by deleting a subset of bins that originally came from $\mathcal{S}_1$. However, note that when we have a problem instance containing many occurrences of items of the same type, there might be many subsets of bins that can be eliminated to get rid of the duplicates. Indeed, these different choices may result in variations to the number of bins that are subsequently inherited from $\mathcal{S}_1$ and also the amount of repair then needed.

Stated more precisely, let $\mathcal{S}^{(t)} = \{S_1^{(t)}, \ldots, S_k^{(t)}\}$ be a multiset of multisets representing a solution $\mathcal{S}$ according to item type $t(\cdot)$. That is, $S_i^{(t)} = \{t(j) : j \in S_i\}, \forall S_i \in \mathcal{S}$. Without loss of generality, now assume that in applying the GGA operator we intend to concatenate the first l bins of a parent solution $\mathcal{S}_2$ to a parent solution $\mathcal{S}_1$. We are now interested in establishing a subset $\mathcal{S}^* \subseteq \mathcal{S}_1^{(t)}$ that *covers* the multiset $\mathcal{U} = \{j : j \in S_{2,i}^{(t)}, 1 \leq i \leq l\}$. This covering $\mathcal{S}^*$ will then specify the sets (bins) that need to be removed from $\mathcal{S}_1$ to cope with the issue of duplicates; additionally, the multiset $\mathcal{U} \cap (\bigcup S_i^*)$ will define the missing items, which will then need to be dealt with by the repair operator.

Note that because $\mathcal{U} \subseteq \bigcup_{i=1}^{k_1} S_{1,i}^{(t)}$, a valid covering is achievable according to the following steps, which are to be repeated until $\mathcal{U} = \emptyset$. To start, let $\mathcal{S}^* = \emptyset$:

1) Choose $S_{1,i}^{(t)} \in \mathcal{S}_1^{(t)}$ for which $S_{1,i}^{(t)} \cap \mathcal{U} \neq \emptyset$.
2) $\mathcal{S}^* \leftarrow \mathcal{S}^* \cup \{S_{1,i}^{(t)}\}$.
3) $\mathcal{U} \leftarrow \mathcal{U} - S_{1,i}^{(t)}$.
4) $\mathcal{S}_1^{(t)} \leftarrow \mathcal{S}_1^{(t)} - \{S_{1,i}^{(t)}\}$.

Using this procedure, a number of different heuristics might be used in Step 1 for influencing the type of covering that is achieved. Most obviously, we might make a random choice, which is what we did to produce the results in Table II. We call this heuristic h_1. A second option, h_2, is to choose the $S_{1,i}^{(t)}$ for which $|S_{1,i}^{(t)} \cap \mathcal{U}|$ is maximised. This operates under the assumption that we are seeking to minimise $|\mathcal{S}^*|$, thereby reducing the number of bins deleted from $\mathcal{S}_1$ and maximising the number of bins that are inherited from the parent solutions. (Note, however, that the task of minimising $|\mathcal{S}^*|$ is actually a generalisation of the NP-hard set covering problem (featuring multisets instead of sets); hence, this heuristic—whose use essentially gives the well-known greedy algorithm for set covering—only determines $|\mathcal{S}^*|$ approximately.) Other heuristic options include choosing the $S_{1,i}^{(t)}$ for which the corresponding bin i has the largest amount of spare capacity (thereby encouraging bins with large amounts of spare capacity to be eliminated) or seeking to maximise $|\mathcal{S}^*|$ in order to encourage large amounts of repair and therefore prolong population diversity. However, these alternatives were not found to improve performance on the whole; consequently, only h_1 and h_2 are considered below.

Note that the same set covering method and heuristics as above can also be used for the two remaining recombination operators. For GPX' the above procedure needs to be executed in each of the $\min(|\mathcal{S}_1|, |\mathcal{S}_2|) - 1$ iterations of the operator. That is, w.l.o.g., each time a bin $S_{2,i} \in \mathcal{S}_2$ is moved into the offspring solution, the procedure is applied using $\mathcal{S}_1^{(t)}$ and $\mathcal{U} = S_{2,i}^{(t)}$. This is also the case for GPX, though slight modifications are also required to cope with the different way in which duplicates are dealt with (see Section IV-B).

Table III shows the results achieved when using heuristics h_1 and h_2 with the GGA and GPX operators. Note that only the "r" instances are considered here (the heuristics are actually equivalent with the "a" instances as they feature no repeated items). We also leave out the GPX' operator as it was always outperformed by GPX, GGA, or both.

The positions of the asterisks in the table indicate that heuristic h_2 consistently improves on the results of the original recombination operators. Thus, there seems to be an advantage in seeking to increase the number of bins (building blocks) inherited from the parents. This stands to reason because, when fewer bins are inherited by an offspring, more bins will need to be formed using the repair procedure; hence these parts will not be subject to the refinements made in previous iterations of the EA.

The locations of the bold text and daggers (†) in Table III also indicate the same patterns as Table II—i.e., that the GGA operator produces the best results overall for small-group instances, with GPX being better for large-group instances. As with Fig. 11, Fig. 13 also shows that these characteristics remain relatively stable throughout the run.

TABLE III

RESULTS ACHIEVED AT THE CUT-OFF USING HEURISTICS h_1 AND h_2 WITH THE GGA AND GPX OPERATORS. THE LOWEST MEAN VALUES ACROSS THE FOUR ALGORITHMS ARE MARKED IN BOLD. FOR EACH RECOMBINATION OPERATOR, ASTERISKS INDICATE STATISTICAL SIGNIFICANCE AT ≤ 0.05 (*) AND ≤ 0.01 (**) AS WITH TABLE I. THE † SYMBOL IS USED IN THE SAME WAY, BUT INDICATES STATISTICAL DIFFERENCE BETWEEN THE BEST RESULT OF EACH RECOMBINATION OPERATOR.

			GGA (h_1)		GGA (h_2)		GPX (h_1)		GPX (h_2)	
Type, $\|U\|$	# Inst.	TMin[a]	Bins[b]	TMin %[c]	Bins	TMin %	Bins	TMin %	Bins	TMin %
orig., r, 100	240	38.66	3.47	19.2	3.47	19.2	3.49	17.9	3.48	18.3
orig., r, 200	240	75.50	6.35	7.1	†**6.33**	7.9	6.42	6.7	**6.37	7.5
orig., r, 300	240	115.10	10.09	2.1	*††**10.07**	2.9	10.26	2.1	**10.15	2.5
orig., r, 400	240	154.44	13.62	1.3	**††**13.59**	2.1	13.95	0.0	**13.75	1.3
orig., r, 500	240	195.03	17.60	1.3	**††**17.55**	1.7	18.02	0.4	**17.75	0.8
half., r, 100	240	20.52	0.22	78.8	0.22	79.2	**0.21**	79.6	0.22	79.2
half., r, 200	240	39.97	0.39	62.9	**0.35	**66.7	0.38	63.8	*0.35	*66.7
half., r, 300	240	60.80	0.76	43.3	**††**0.67**	**50.8	0.78	43.8	**0.71	**48.3
half., r, 400	240	81.34	0.86	35.0	**††**0.74**	**44.2	0.95	31.7	**0.80	**41.7
half., r, 500	240	101.75	1.20	22.9	**††**1.03**	**31.3	1.34	22.1	**1.14	**29.2
quar., r, 100	240	11.48	0.07	92.9	0.07	92.9	0.07	93.3	**0.06**	94.2
quar., r, 200	240	22.14	0.12	88.3	**0.09	*91.3	0.09	91.3	**††**0.05**	**††94.6
quar., r, 300	240	33.62	0.17	82.9	*0.15	85.0	0.15	85.4	*††**0.12**	**†87.9
quar., r, 400	240	44.80	0.22	78.8	**0.17	**83.8	0.18	82.1	**†**0.15**	**85.8
quar., r, 500	240	56.05	0.33	67.5	**0.24	**75.8	0.30	70.4	****0.23**	**77.1

[a]TMin $= \lceil (\sum_{i \in U} A(i))/HW \rceil$ (mean across all instances).
[b]Number of bins beyond TMin (mean across all instances).
[c]Percentage of instances where a solution using TMin bins was found.

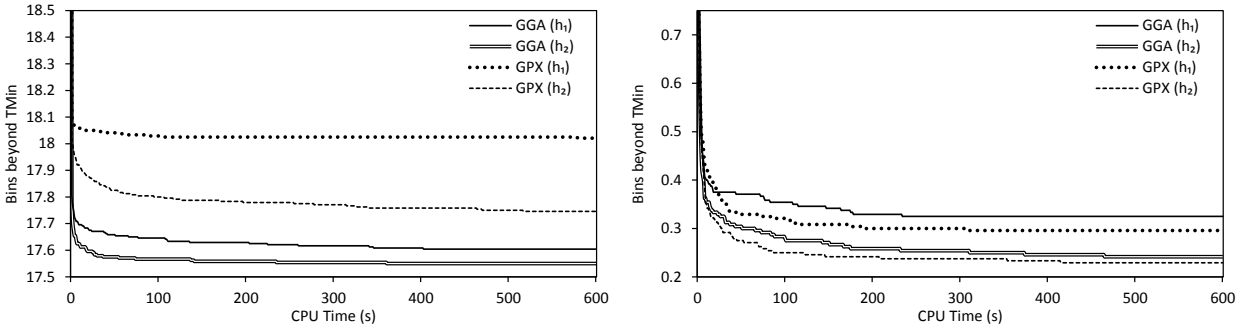

Fig. 13. Run profiles using the $|U| = 500$ original "r" and quarter "r" instances respectively. Each line is the average across all 240 instances at each second.

A third possibility for improving the results of our EA is to generalise the GPX operator so that $q \geq 2$ parents contribute towards the production of each offspring [27]. Here, our suggested multi-parent operator produces offspring in the same manner as GPX except that, in each iteration, the fullest bin from across *multiple parents* is chosen to be moved into the offspring. The intention behind this increased choice is that fuller bins will be identified, hopefully resulting in a higher quality offspring solution once all of its $\min\{|\mathcal{S}_j| : 1 \leq j \leq q\} - 1$ bins have been constructed. To prohibit too many bins being inherited from one particular parent, a rule can also be specified with this operator specifying that if the ith bin in an offspring has originated from parent $\mathcal{S}_j$, then no further bins should be taken from this parent until a certain number of bins have then been taken from other parents. In our case we set this value to be $q/2$. Note in particular that GPX is therefore an application of the multi-parent operator using $q = 2$.

To test this new operator we repeated our previous trials on all problem instances using heuristic h_2 together with the following settings for population size and q: 10, 2; 10, 4; 10, 8; 20, 2; 20, 4; 20, 8; and 20, 16. For the original and half instances we found that the first of these settings offered the best results, with larger values for q tending to degrade performance. Such findings are consistent with those noted in Section IV-D, namely that for these problems too much bias is being placed on inheriting the fullest bins as opposed to encouraging a wider sampling of the solution space. On the other hand, as also seen in Section IV-D, such bias is seen to be less problematic with the quarter instances. Indeed, for the 260 problem instances for which $|U| = 500$, values of $q = 4$ and 6 resulted in solutions requiring 0.027 and 0.023 fewer bins on average[7] compared to $q = 2$. However, such improvements were not observed with the other problem sizes in this class.

V. CONCLUSIONS

This paper has described an exact polynomial-time algorithm for the pair sequencing problem, which has then been used to solve the related trapezoid packing sub-problem. We have also seen how this EULER-SPLICE method can be combined with specialised evolutionary and local search methods to produce high-quality results for the more general trapezoid packing problem. Of course, EULER-SPLICE might also be used with other bin packing methods. For example, it could be used in conjunction with column generation techniques to generate a large pool of feasible packings, a subset of which

[7]Using populations of size 10. Both differences were statistically significant at the $p \leq 0.025$ level.

TABLE IV
PERCENTAGE OF THE 260 LARGE PROBLEM INSTANCES THAT ARE "SOLVED" AT VARIOUS TIME LIMITS FOR THE BEST PERFORMING ALGORITHMS.

Method & Instances	5s	10s	30s	60s	300s	600s
GGA (h_2), orig., $\|U\| = 500$	88.85	91.92	96.15	97.31	99.23	99.62
GGA (h_2), half, $\|U\| = 500$	71.15	77.31	84.62	86.15	92.69	94.62
GPX (h_2), quart., $\|U\| = 500$	75.00	82.69	91.54	95.00	99.23	99.23

might then be selected to form an optimal solution [28]. The EA itself might also be used to generate such a pool by collecting information on good packings of items during a run. This pool could then be used with an integer programming formulation of the set covering problem as an additional post-optimisation phase to try to make further improvements to a solution—see, for example, the approach of Malaguti et al. [29] for the related graph colouring problem.

In our evolutionary algorithm we have observed that recombination operators showing less bias towards well-filled bins seem better suited to problem instances where the number of items per bin is small, whereas the opposite is true for instances with many items per bin. We have also demonstrated the advantages of using set-covering heuristics for encouraging fewer bins to be destroyed during recombination, helping to increase the amount of information passed from parents to the offspring.

Throughout this paper we have used a fixed run-time limit of 600s in our trials; however, as Figs. 11 and 13 have illustrated, the majority of improvements are achieved in very early stages of runs. To demonstrate this further, Table IV reports the quality of results achieved at different times using the best reported algorithms on the largest problem instances of each class. (Note that in this table an instance is considered "solved" by the corresponding method if its solution uses the same number of bins as the best observed value for this instance from across all of our trials—it does *not* imply optimality as such.) We see that even for these large problem instances, over three quarters have been "solved" in less than ten seconds.

A further practical application of the TPP is in the laying of decked flooring, where it is often preferable for decking boards to be laid diagonally across floor joists. Often, areas of floor will be square or rectangular in shape with boards being laid at a 45° angle; hence the corresponding cutting problem will actually be a special case of the TPP for which exact polynomial-time algorithms could be available.

Another bin packing variant with a sub-problem related to the TPSP is the *box cutting problem*, first defined by Goulimis [30]. This problem models the task of using a specialised machine to cut fixed-height rectangular cardboard items from larger strips of cardboard so that the number of larger strips (bins) used is minimised. Like the TPP, each item in this problem features projections on its left- and right-hand sides, in this case defining where the machine is to score the cardboard ready for folding. However, due to the mechanics of this machine, the scoring points on adjacent items within a bin must be a certain minimum distance apart. As with Definition 1 the resultant sub-problem can therefore be defined using a set of unordered pairs of nonnegative integers $\mathcal{P}$. The task is to then seek an ordering of the elements $\mathcal{X}$ such that $\mathbf{rhs}(i) + \mathbf{lhs}(i+1) \geq C, \forall i \in \{1, \ldots, n-1\}$ for some given constant C. Heuristics for this sub-problem have previously been suggested by Lewis et al. [1] and Becker et al. [31]; however, at the time of writing we are not aware of a proof of this sub-problem's NP-completeness.

ACKNOWLEDGEMENT

The authors would like to thank Phil Dumbreck and Danny Groves for useful discussions on the PSP in initial stages of this research.

REFERENCES

[1] R. Lewis, X. Song, K. Dowsland, and J. Thompson, "An investigation into two bin packing problems with ordering and orientation implications," *European Journal of Operational Research*, vol. 213, pp. 52–65, 2011.
[2] J. Edmonds and E. Johnson, "Matching, Euler tours, and the Chinese postman," *Mathematical Programming*, vol. 5, pp. 88–124, 1973.
[3] J. Lenstra and A. Rinnoy-Kan, "On general routing problems," *Networks*, vol. 6, pp. 273–280, 1976.
[4] L. Euler, "Solutio problematis ad geometriam situs pertinentis," *Commentarii Academiae Scientiarum Petropolitanae*, vol. 8, pp. 128–140, 1741.
[5] C. Hierholzer and C. Wiener, "Ueber die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren," *Mathematische Annalen*, vol. 6, no. 1, pp. 30–32, 1873.
[6] H. Fleischner, *Eulerian Graphs and Related Topics: Part 1*, ch. X.1 Algorithms for Eulerian Trails, pp. 1–13. Elsevier, 1991.
[7] J. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, pp. 48–50, 1956.
[8] E. Falkenauer, *Genetic Algorithms and Grouping Problems*. John Wiley and Sons, 1998.
[9] R. Lewis, "A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing," *Computers and Operations Research*, vol. 36, no. 7, pp. 2295–2310, 2009.
[10] S. Martello and P. Toth, "Lower bounds and reduction procedures for the bin packing problem," *Discrete Applied Mathematics*, vol. 28, pp. 59–70, 1990.
[11] J. Levine and F. Ducatelle, "Ant colony optimisation and local search for bin packing and cutting stock problems.," *Journal of the Operational Research Society*, vol. 55(12), no. 7, pp. 705–716, 2003.
[12] A. Yao, "New algorithms for bin packing," *J. ACM*, vol. 27, no. 2, pp. 207–227, 1980.
[13] M. Garey, R. Graham, D. Johnson, and A. Yao, "Resource constrained scheduling as generalized bin packing," *J. Combinatorial Theoty Ser. A*, vol. 21, pp. 257–298, 1976.
[14] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham, "Worst-case perfromance bounds for simple one-dimensional packing algorithms," *SIAM Journal of Computing*, vol. 3, pp. 299–325, 1974.
[15] M. Garey and D. Johnson, "A 71/60 theorem for bin packing," *Journal of Complexity*, vol. 1, no. 1, pp. 65–106, 1985.
[16] "http://www.rhydlewis.eu/resources/trapboxprobs.zip."
[17] R. Lewis, J. Thompson, C. Mumford, and J. Gillard, "A wide-ranging computational comparison of high-performance graph colouring algorithms," *Computers and Operations Research*, vol. 39, no. 9, pp. 1933–1950, 2012.
[18] R. Lewis, *Springer Handbook of Computational Intelligence*, ch. Graph Coloring and Recombination, pp. 1239–1254. Studies in Computational Intelligence, Springer, 2015.
[19] M. Quiroz-Castellanos, L. Cruz-Reyes, J. Torres-Jimanez, C. Gómez, H. Huacuja, and A. Alvim, "A grouping genetic algorithm with controlled gene transmission for the bin packing problem," *Computers and Operations Research*, vol. 55, pp. 52–64, 2015.
[20] A. Singh and A. Gupta, "Two heuristics for the one-dimensional bin-packing problem," *OR Spectrum*, vol. 29, pp. 765–781, 2007.
[21] P. Galinier and J.-K. Hao, "Hybrid evolutionary algorithms for graph coloring," *Journal of Combinatorial Optimization*, vol. 3, pp. 379–397, 1999.

[22] R. Lewis and E. Pullin, "Revisiting the restricted growth function genetic algorithm for grouping problems," *Evolutionary Computation*, vol. 19, no. 4, pp. 693–704, 2011.
[23] D. Porumbel, J.-K. Hao, and P. Kuntz, "An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring," *Computers and operations research*, vol. 37, pp. 1822–1832, 2010.
[24] C. Mattiussi, M. Waibel, and D. Floreano, "Measures of diversity for populations and distances between individuals with highly reorganizable genomes," *Evolutionary Computation*, vol. 12, no. 4, pp. 495–515, 2004.
[25] E. Coll, G. Duran, and P. Moscato, "A discussion on some design principles for efficient crossover operators for graph coloring problems," *Anais do XXVII Simposio Brasileiro de Pesquisa Operacional, Vitoria-Brazil*, 1995.
[26] R. Lewis and B. Paechter, "Finding feasible timetables using group based operators," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 3, pp. 397–413, 2007.
[27] Z. Lü and J.-K. Hao, "A memetic algorithm for graph coloring," *European Journal of Operational Research*, vol. 203, no. 1, pp. 241 – 250, 2010.
[28] J. Valério de Carvalho, "Exact solution of bin-packing problems using column generation and branch-and-bound," *Annals of Operations Research*, vol. 86, pp. 629–659, 1999.
[29] E. Malaguti, M. Monaci, and P. Toth, "A metaheuristic approach for the vertex coloring problem," *INFORMS Journal on Computing*, vol. 20, no. 2, pp. 302–316, 2008.
[30] C. Goulimis, "Minimum score separation—an open combinatorial problem associated with the cutting stock problem," *Journal of the Operational Research Society*, vol. 55, pp. 1367–1368, 2004.
[31] K. Becker and A. Gutam, "A heuristic for the minimum score separation problem—a combinatorial problem associated with the cutting stock problem," *Journal of the Operational Research Society*, vol. 66, no. 1297-1311, 2015.

Rhyd Lewis is a senior lecturer in operational research at the School of Mathematics, Cardiff University, Wales. He holds a PhD in computing and operational research (Edinburgh Napier University, 2006) and is the author of the book *Graph Colouring: Algorithms and Applications* (Springer, 2015). Further publications are listed at his website www.RhydLewis.eu.

Penny Holborn is a lecturer in mathematics, operational research and statistics at the University of South Wales. Her PhD (Cardiff University, 2013) applied heuristic and metaheuristic techniques to dynamic vehicle routing problems. She was employed as a research associate from 2013–2015, where she used a range of operational research techniques to help solve a variety of real world problems as part of a Healthcare Modelling Unit for the Aneurin Bevan University Health Board in Wales.