



Exact algorithms in bar nesting: How to cut general items from linear stocks so that wastage is minimised

Rhyd Lewis ^a,* Louis Bonnet ^b

^a School of Mathematics, Cardiff University, CF24 4AX, Wales, United Kingdom

^b Laboratory for Analysis and Architecture of Systems, CNRS, Toulouse 31031, France

ARTICLE INFO

Dataset link: <https://doi.org/10.5281/zenodo.11657149>

Keywords:

Manufacturing and logistics
Bar nesting
Stock cutting
Bin packing
Graph theory

ABSTRACT

This paper proposes exact, polynomial-time algorithms that solve the problem of cutting items with angled sides from a single linear stock so that wastage is minimised. In industry, this problem is called “bar nesting”. Here we give an algorithmic framework that solves several important variants of the problem, including cutting items from stocks with asymmetric cross-sections, cutting items whose sides occur on different planes, and the minimum score separation problem.

1. Introduction

In manufacturing and construction industries, bar nesting software is often used to help optimise the cutting patterns of linear materials such as reinforcing bars, girders, pipes, wooden joists, and window frames. An important objective in such software is to determine arrangements of the desired items on long stocks of the appropriate material so that, when cut from these stocks, resultant wastage is minimised. Several companies currently offer bar nesting software for industrial use, often as part of a broader suite of construction-related CAD tools. Popular vendors include AutoRebar, RebarCAD, AutoBar-Sizer, and TopSolid Design. The third of these tools uses algorithms previously developed by the first author of this paper (Lewis & Holborn, 2017), while contributions to TopSolid’s bar nesting functionality have been made by the second author, who was previously employed by the company.

The standard industrial process of cutting items from linear stocks is illustrated in Fig. 1(a). Here, the stock (bar) is fed through a device that allows a 360° rotation along its length, as indicated. Once the stock is in position, a straight cut is then performed vertically downwards at the desired location. The cutting blade can also be rotated through 360° on the axis perpendicular to the stock, allowing angled cuts to be made.

Because stocks can be rotated between individual cuts, this allows the angled ends of items to occur on different planes. Fig. 1(b) illustrates why this is desirable, as it allows sets of items to be cut that, when joined, can form elaborate 3D structures. The cross sections of individual lengths of stock can also vary. Examples are shown in

Fig. 1(c), including hollow tubes, L- and I-shaped girders, and window seals.

The standard industry practice is for linear stocks to be manufactured in fixed lengths, from which the desired items are then cut. To minimise wastage in the production of a set of items, it is therefore necessary to partition the required items into subsets, with each subset being assigned to a separate length of stock, such that (1) all items assigned to a subset can be cut from an individual stock, and (2) the number of subsets (stocks required) is minimised. The combination of these two requirements makes the bar nesting problem a member of a larger set of cutting and packing problems that Wäscher et al. (2007) call “fixed-dimension input-minimisation problems”. Perhaps the most widely known member of this class is the famous one-dimensional bin packing problem (BPP), which involves taking a set of items of differing lengths and “packing” them into a minimum number of fixed-length “bins” so that the sum of item lengths in each bin does not exceed the bin length. Garey and Johnson (1979) have shown that the BPP is \mathcal{NP} -hard, which has led to the proposal of many different techniques for the problem, including approximation algorithms, heuristics, and exponential-time exact techniques (Coffman et al., 2013; Falkenauer, 1998; Garey & Johnson, 1981).

Despite the difficulty of solving the BPP, the subproblem of deciding whether a particular set of items can be packed into a *single* bin is trivial: we simply check that the sum of their lengths does not exceed the length of the bin. The commutativity of this summing operation also

* Corresponding author.

E-mail addresses: lewisr9@cardiff.ac.uk (R. Lewis), lbonnet@laas.fr (L. Bonnet).

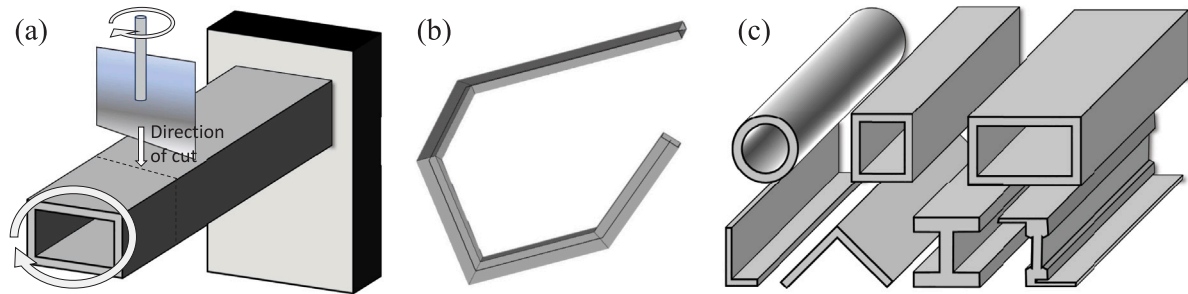


Fig. 1. Part (a) shows how the stock and blade can be rotated during the cutting process. Part (b) shows a structure formed by joining items whose ends have been cut on different planes. Part (c) shows cross-sections of different types of stock materials.

implies that the *order* in which items occur in the bin is unimportant. Note, then, that the BPP can be considered a special case of the bar nesting problem in which (a) all cuts are made at right angles to the length of the stock, and (b) all cuts occur on the same plane (that is, stocks are not rotated between cuts). This implies that the bar nesting problem is also \mathcal{NP} -hard in general.

As we have noted, industrial applications of bar nesting problems involve the production of items with angled ends that can occur on different planes. Unlike the BPP, the way in which items are cut from individual stocks (or, equivalently, packed into individual bins), can therefore affect the amount of material wastage incurred. In this paper we will specifically consider this latter feature, focusing on efficient methods for arranging (and cutting) items on a *single* linear stock. More specifically, given a set of items whose total length is seen to not exceed the stock length, we will consider polynomial-time exact methods that determine whether or not the items can be feasibly cut from this stock. Such methods might then be used in conjunction with a broader algorithm for tackling the multiple-stock version of this problem.

2. Literature review and contributions

In this section, we focus on cutting and packing problems specifically related to bar nesting. Each of these problems can be considered one-dimensional in that, for a given set of items to be feasible, their total length should not exceed the given stock length. As we have seen, for the BPP the satisfaction of this single constraint is sufficient; however, in the following problems, feasibility also depends on factors including the ordering, alignment, and compatibility of items. Two tasks can therefore be identified: (a) the single-stock (decision) subproblem, where we are interested in determining whether a set of items can be cut from a single stock (or, equivalently, packed into a single bin), and (b) the multi-stock (optimisation) problem, where we want to determine the minimum number of stocks needed to cut/pack a set of items.

One of the first considered problems of this type was studied by Jansen (1999). In this work, the BPP is extended by associating each item $i \in I$ with a vertex v_i in a simple graph $G = (V, E)$. An edge $\{v_i, v_j\} \in E$ then signifies that the pair of items i, j are *incompatible* and cannot be assigned to the same stock, even if the length of the stock allows it. If $E = \emptyset$, then this problem is equivalent to the BPP; otherwise, it is an extension of the \mathcal{NP} -hard graph colouring problem (Lewis, 2021). Note that the single-stock subproblem is easy to solve in this case—in addition to summing the lengths of the items, we simply need to confirm that $\{v_i, v_j\} \notin E$ for all pairs of items i, j on the stock. Jansen (1999) also suggests an algorithm for the multi-stock variant of this problem that gives asymptotic approximations for several graph topologies including trees, grid graphs, and planar graphs.

Shachnai and Tamir (2001) have studied a different extension to the BPP in which each item i is preallocated a colour $c(i) \in \mathbb{N}$. Each stock then has an upper bound k on the number of different colours it can accommodate. Solving the single-stock subproblem is again easy in this case: given a set of items $I = \{i_1, \dots, i_n\}$ whose total length does not exceed the stock length, we simply determine the set of colours used by these items, $C = \{c(i) : i \in I\}$, and check that $|C| \leq k$. A second problem variant is also considered by Shachnai and Tamir (2001) in which each item i has a choice of which colour to assume, giving $c(i) \subseteq \mathbb{N}$. In this case, solving the single-stock subproblem is equivalent to the task of finding a set cover of size k (or less) for the universe I using the family of subsets $\{\{i : i \in I \wedge j \in c(i)\} : j \in \mathbb{N}\}$. The set cover problem is known to be \mathcal{NP} -complete, however (Garey & Johnson, 1979).

Another variant of the BPP in which items are preallocated colours is considered by Borges et al. (2024). In this problem, items on the same stock cannot be placed in adjacent positions when they have the same colour. The single-stock subproblem therefore involves determining a sequence $\langle i_1, \dots, i_n \rangle$ of the given items such that $c(i_j) \neq c(i_{j+1}) \forall j \in \{1, \dots, n-1\}$. Borges et al. (2024) show that such a sequence will exist whenever $\lceil n/2 \rceil$ or fewer items have the same colour. They also give an $\mathcal{O}(n)$ algorithm for producing such sequences, which they then combine with other operators to tackle the multi-stock problem variant.

In other work, Goulimis (2004) considers an extension to the BPP that arises in the manufacture of cardboard boxes. In this problem, each item $i \in I$ has a length l_i and, in addition, is marked with two vertical score lines in predetermined places. The distances between each score line and the nearest end of the item are then known as the *score lengths* x_i and y_i (where $x_i + y_i < l_i$). In the described industrial process, when a cut is performed on the stock, two scores are made simultaneously on the items on either side at the prescribed distances; however, because of the way cuts are performed, these scores cannot be less than τ distance units apart, making some arrangements of items infeasible.

More formally, the single-stock subproblem of Goulimis (2004) involves taking a set of items $I = \{i_1, \dots, i_n\}$ whose combined length $\sum_{i \in I} l_i$ does not exceed the stock length. From this, the multiset $P = \{\{x_i, y_i\} : i \in I\}$ is formed, and the task is to determine a sequence of the elements of P in which each item is an ordered pair such that the sum of adjacent values from neighbouring ordered pairs is not less than τ . For example, using the input $P = \{\{1, 2\}, \{1, 4\}, \{2, 3\}, \{2, 4\}\}$ with $\tau = 5$, a feasible solution is $\langle (2, 3), (2, 4), (1, 2), (4, 1) \rangle$, whereas the solution $\langle (3, 2), (2, 4), (1, 2), (4, 1) \rangle$ is infeasible (the adjacent values between the first and second elements sum to a value less than τ). In the literature, the above single-stock subproblem is known as the *minimum score separation problem*. Lewis et al. (2011) proposed a greedy heuristic for this problem, though it was proved to be polynomially solvable by Becker (2010) using the concept of alternating Hamiltonian paths. More recently, Hawa et al. (2022) extended these ideas to give

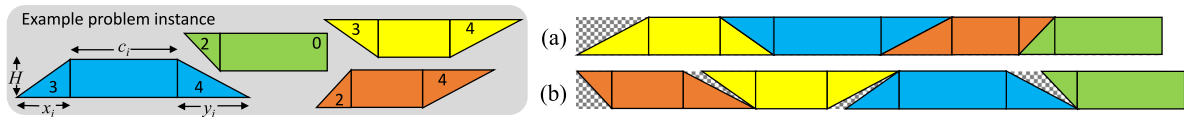


Fig. 2. A small trapezoid packing problem instance involving four items. Part (a) shows an optimal arrangement when both left- and up-rotations are permitted; Part (b) shows the optimum solution when only up-rotations are permitted.

an exact $\mathcal{O}(n^2)$ algorithm for the problem. They then combined this with heuristic operators to tackle the multi-stock variant.

Garraffa et al. (2016) have also considered a BPP in which the order and alignment of items on a stock are important. In their case, material wastage is said to occur between each pair of adjacent items i, j on a stock, defined by a value $w(i, j)$. The single-stock subproblem then involves determining a sequence $\langle i_1, \dots, i_n \rangle$ of the given items such that the combined length and wastage, $\sum_{j=1}^n l_{i_j} + \sum_{j=1}^{n-1} w(i_j, i_{j+1})$, does not exceed the stock length. In their case, values for $w(i, j)$ are defined using an arbitrarily filled $n \times n$ matrix, so solving the single-stock subproblem is equivalent to the \mathcal{NP} -complete travelling salesman problem (TSP). In addition, Garraffa et al. (2016) note that the multi-stock variant is equivalent to the \mathcal{NP} -hard distance-constrained vehicle routing problem.

A problem related to this was previously considered by Lewis et al. (2011), who examined the problem of cutting trapezoidal items from stocks. This problem originated in the roofing industry, where roof trusses need to be cut from long pieces of timber. Here, all items have the same height, but their lengths and end angles can vary. The single-stock subproblem then involves determining a sequence and orientation of the given trapezoids so that the inter-item wastage is minimised. If this inter-item wastage plus the total area of the items is seen to be less than the area of the stock, then an arrangement exists that allows all items to be cut from the stock.

A small example of this trapezoid packing problem (TPP) is shown in Fig. 2(a). Note that items can be left-rotated and/or up-rotated in this problem. In Part (a) of the figure, for example, the yellow item has been both left- and up-rotated from its original state. This leads to a choice of four different orientations for each item. Importantly, the use of up-rotations also allows adjacent angled ends to be nested, ensuring that “/” angles are always adjacent to other “/” angles, and “\” angles are always adjacent to other “\” angles. Lewis et al. (2011) made use of greedy heuristics and integer programming for the single-stock TPP, though Lewis and Holborn (2017) later found the problem to be polynomial-time solvable using their $\mathcal{O}(n^3)$ EULER-SPLICE algorithm. The latter paper also suggested several ways of combining EULER-SPLICE with specialised heuristics for tackling the multi-stock version of the problem.

A single-stock problem related to the TPP is also considered by Gilmore and Gomory (1964). Here, a problem instance is defined by a multiset of ordered integer pairs $P = \{(x_i, y_i) : i \in I\}$, and the aim is to determine a sequence of these elements such that the sum of the costs occurring between adjacent pairs is minimised. Although this problem is \mathcal{NP} -hard under an arbitrary cost function, Gilmore and Gomory (1964) give an exact quadratic-time algorithm for cases where the cost between neighbouring values y and x in a solution can be written as

$$\begin{cases} \int_y^x g_1(z) dz & \text{if } x \geq y \\ \int_x^y g_2(z) dz & \text{otherwise} \end{cases} \quad (1)$$

where $g_1(z)$ and $g_2(z)$ are any integrable functions satisfying

$$g_1(z) + g_2(z) \geq 0. \quad (2)$$

Note that this algorithm does not support the rotation of items; however, it is still suitable for solving a variant of the TPP considered in Section 3.2 later.

2.1. Contributions and paper plan

The next section of this paper begins with a more detailed description of the (single-stock) trapezoid packing problem (TPP) and shows how and when it can be used in industrial bar nesting applications. We then show how the EULER-SPLICE method for this problem can be (a) modified to feature an improved complexity of $\mathcal{O}(n^2)$, and also (b) adapted to solve a more restricted version of the TPP in which items cannot be left-rotated. Formal proofs of correctness, which have hitherto not been given for the EULER-SPLICE method, are presented in Section 3.3.

In Section 4 we then expand significantly on this work by using our previous method of proof to define a generalised polynomial-time algorithm that, for the first time, exactly solves several other problem variants arising in industrial bar nesting applications. These include situations involving angled cuts on bars with asymmetric cross-sections, cuts occurring on different planes, and also the minimum score separation problem described earlier. Section 5 presents an empirical analysis of these algorithms over a large set of problem instances while, finally, Section 6 concludes the work and makes suggestions for future research.

3. Euler-Splice and bar nesting

As noted in the previous section, the TPP involves taking a set of fixed-height trapezoidal items and arranging them onto a single stock (bar) such that inter-item wastage is minimised. More formally, we are given a set of items $I = \{i_1, \dots, i_n\}$ in which each item i has the same height H , a “central length” c_i , and two “projection lengths” x_i and y_i that define the angles of its lateral sides (see Fig. 2). The area of an item i is denoted by $A(i) = Hc_i + \frac{1}{2}Hx_i + \frac{1}{2}Hy_i$, and the problem of deciding whether the items of I can be cut from a single $H \times L$ stock now involves determining an arrangement of the items such that the inter-item wastage is less than $HL - A(I)$, where $A(I) = \sum_{i \in I} A(i)$.

Note that, because the trapezoidal items can be up-rotated, adjacent projections can always be nested in the manner described previously. This is due to the theorem of Lewis et al. (2011), which we reproduce here.

Theorem 1 (Lewis et al., 2011). *When minimising wastage between successive trapezoids in a defined sequence, we only need to decide whether each trapezoid should be left-rotated.*

Proof. Consider a set of trapezoidal items arranged 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, “/” angles will be adjacent to other “/” angles and “\” will be adjacent to other “\” angles). Now suppose the contrary, and that two adjacent trapezoids in this arrangement do not nest. If we take all trapezoids to the right of this join and perform an up-rotation, this join will be nested, decreasing the wastage, and leaving the remaining joins in the sequence unchanged. Hence, the original orientation did not give the minimal wastage. \square

This nesting property implies that, w.l.o.g., the area of inter-item wastage between any two adjacent projections x, y can be calculated as $\frac{1}{2}H(\|x - y\|)$. Since H is a constant, this can be further simplified to $\|x - y\|$. The task of arranging the items of I therefore involves determining an ordering of the n items and, for each item i , deciding whether the projection x_i or y_i should occur on the left. This can be restated as follows.

Definition 1 (*The Pair Sequencing Problem (PSP) of Lewis & Holborn, 2017*). Given a set of items $I = \{i_1, \dots, i_n\}$, let P be the multiset of unordered pairs containing their projection lengths, $P = \{\{x_i, y_i\} : i \in I\}$, and let S be an ordering of the elements of P in which each element is expressed as an ordered pair. The PSP involves identifying the solution S that minimises the cost function:

$$\mathcal{F}(S) = \left(\sum_{j=1}^{|S|-1} f(\text{rhs}(j), \text{lhs}(j+1)) \right) + f(\text{rhs}(|S|), \text{lhs}(1)) \quad (3)$$

where $\text{lhs}(j)$ and $\text{rhs}(j)$ give the values of the left- and right-hand sides of the j th ordered pair in S , and where $f(x, y) = \|x - y\|$ gives the difference between the two values x and y .

To account for the triangles of wastage occurring on the left and right of the stock, it is sufficient to add a single dummy element $\{0, 0\}$ to P in the above definition. An optimal solution S for this PSP corresponds to an arrangement of the set I of trapezoidal items that minimises inter-item wastage. If this wastage is less than $HL - A(I)$, then the single-stock (decision) problem is answered in the affirmative (i.e., all items of I can be cut from a single $H \times L$ stock). As an example, the set of items in Fig. 2 gives the problem $P = \{\{3, 4\}, \{2, 0\}, \{2, 4\}, \{3, 4\}\} \cup \{\{0, 0\}\}$, which features an optimal solution $S = \langle (0, 0), (4, 3), (3, 4), (4, 2), (2, 0) \rangle$ of cost $\mathcal{F}(S) = 4$, as illustrated in Fig. 2(a).

Recall from Section 1 that, in the described industrial bar cutting process, cuts are made using a blade that moves vertically downward through the stock. If all cuts on a stock are to be made on the same plane, then it is clear that each item can be considered a two-dimensional trapezoid. Moreover, if, on this plane, the stock material exhibits a vertical line of reflectional symmetry or 180° rotational symmetry, then, where necessary, the stock can be rotated by 180° between successive cuts to allow adjacent projections to be nested. (This symmetry exists, for example, in all of the stock materials and planes shown in Fig. 1(c), except for the asymmetric L-shaped bar at the bottom left.) Such an action is equivalent to performing an up-rotation of an item. Consequently, a solution to the PSP will give an optimal way of cutting items from the bar/stock.

The PSP, in effect, allows items to be both left-rotated and up-rotated. However, certain industrial applications may not allow left rotations; for example, if the stock is a bar magnet, or a pipe with a prescribed direction of flow. In these cases, a more restricted version of the PSP is required in which, for each item $i \in I$, the projection x_i is always kept on the left. This gives a second problem definition.

Definition 2 (*The Ordered Pair Sequencing Problem (OPSP)*). Given a set of items $I = \{i_1, \dots, i_n\}$, let P be the multiset of ordered pairs containing their projection lengths, $P = \{(x_i, y_i) : i \in I\}$, and let S be an ordering of the elements of P . The OPSP involves identifying the solution S that minimises the same cost function as Definition 1.

Fig. 2, for example, gives the OPSP instance $P = \{(3, 4), (2, 0), (2, 4), (3, 4)\} \cup \{(0, 0)\}$. This features the optimal solution $S = \langle (0, 0), (2, 4), (3, 4), (3, 4), (2, 0) \rangle$ that has $\mathcal{F}(S) = 6$, as shown in Fig. 2(b). An adaptation of the EULER-SPLICE method for the OPSP will be described in Section 3.2.

Algorithm 1: EULER-SPLICE method for the PSP.

input : A problem instance P .
output : A single optimal solution.

- 1 Let $G = (V, E)$ be the weighted multigraph constructed according to Definition 3. Set L as an empty list.
- 2 **foreach** $i \in \{1, \dots, |V|\}$ **do**
- 3 **if** $\text{deg}(v_i)$ is odd **then** append v_i to L .
- 4 **foreach** $i \in \{1, 3, 5, \dots, |L| - 1\}$ **do**
- 5 Let $u = L[i]$ and $v = L[i + 1]$.
- 6 Add a dummy edge $\{u, v\}$ to G and set $w(u, v) = f(u(u), u(v))$
- 7 Let C_1, C_2, \dots, C_l be the connected (Eulerian) components of G .
- 8 **foreach** $i \in \{1, \dots, l\}$ **do**
- 9 Use Hierholzer's algorithm to generate an Eulerian circuit S_i from C_i .
- 10 Delete any dummy edges from S_i and then replace each vertex v in S_i with its corresponding weight $w(v)$.
- 11 **if** $l \geq 2$ **then** $S_1 = \text{SPLICE}(\{S_1, \dots, S_l\})$
- 12 **return** S_1

3.1. Improving EULER-SPLICE

A pseudocode description of our improved EULER-SPLICE algorithm for the PSP is given in Algorithm 1. A worked example is also shown in Fig. 3(a). At the start of the algorithm, a graph G is first constructed according to the following definition.

Definition 3. Let $G = (V, E)$ be an edge-weighted multigraph with edge set $E = P$. Consequently, the vertex set V has one vertex for each different value occurring in P . In this graph, let the weight $w(v)$ of a vertex $v \in V$ correspond to its numerical value in P ; hence, $\text{deg}(v)$ gives the number of occurrences of the value $w(v)$ in P . For convenience, also assume that vertices are labelled such that $w(v_1) < w(v_2) < \dots < w(v_{|V|})$. Finally, set the weight of each edge $w(u, v) = 0$.

In Lines 2–6 of Algorithm 1, a set of “dummy edges” is added to G to make the degrees of all vertices even. By definition, this makes G a (possibly disconnected) Eulerian graph. Each of the added dummy edges $\{v_i, v_j\}$ will also feature a (positive) weight $w(v_i, v_j) = f(w(v_i), w(v_j))$ that will contribute to the overall cost of the final solution. The purpose of these steps is to therefore identify an appropriate set of dummy edges whose total cost is minimal. (See Theorem 2.)

On completion of these steps, the graph G (including the dummy edges) will comprise $l \geq 1$ Eulerian components C_1, C_2, \dots, C_l . As shown in the remaining steps of Algorithm 1, each component C_i is now converted back to a corresponding subsolution S_i by (a) writing out a sequence of edges corresponding to an Eulerian circuit of C_i , (b) removing the dummy edges from this sequence, and (c) replacing each vertex in the sequence with its corresponding weight. (See Steps 2 and 3 of Fig. 3(a) for an example.) Note that the cost of each subsolution S_i equals the total weight of the dummy edges present in C_i .

In cases where $l = 1$, the optimal solution to problem P will now have been found and the algorithm can terminate (see Theorem 3). Otherwise, the various subsolutions must be *spliced* together to form a single optimal solution. To describe this splicing process, consider the case where we have a pair of subsolutions $S_i = \langle (x_i \dots y_i) \rangle$ and $S_j = \langle (x_j \dots y_j) \rangle$ whose outer values are as indicated. If we now splice these subsolutions by appending one to the other, the result is a new subsolution $\langle (x_i, \dots, y_i)(x_j \dots y_j) \rangle$ that incurs an additional cost of:

$$f(y_i, x_j) + f(y_j, x_i) - f(y_i, x_i) - f(y_j, x_j). \quad (4)$$

Note also that the costs of individual subsolutions do not change under cyclic shifts and inversions. For example, the subsolution $S_i =$

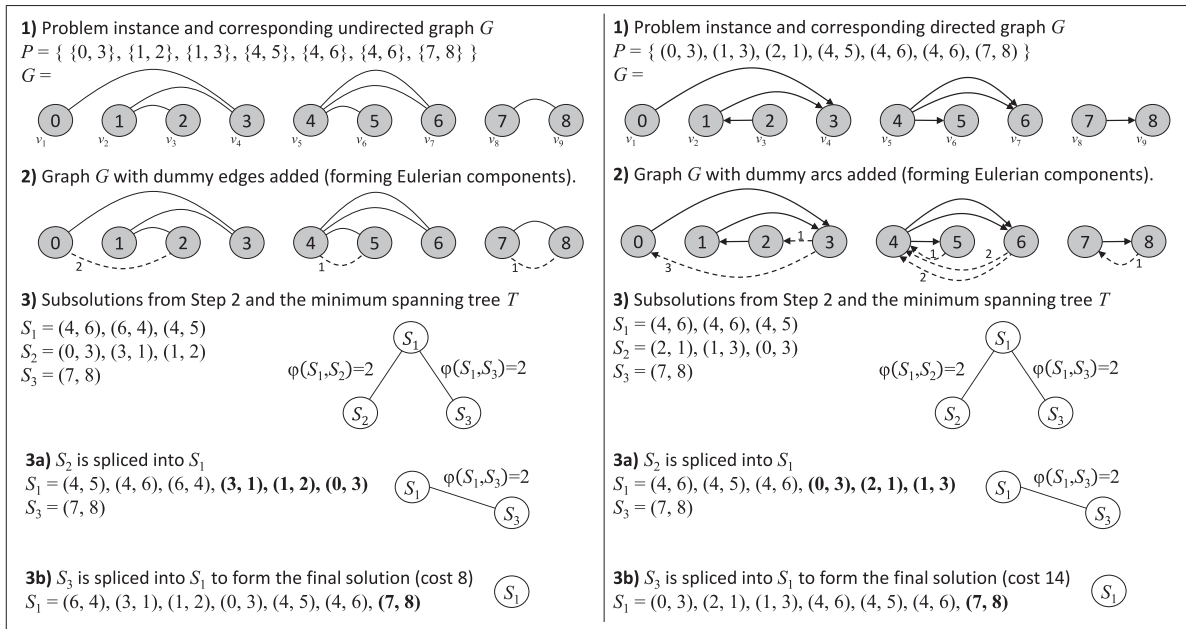


Fig. 3. Example applications of the EULER-SPLICE algorithm for the PSP (left) and OPSP (right). In Steps 1 and 2, the values inside the grey vertices correspond to the vertex weights $w(v)$. The weights of the dummy edges and dummy arcs (dashed lines and arrows) are also indicated.

$\langle(2, 1), (1, 3), (3, 1)\rangle$ (with a cost of one), is equivalent to both $\langle(1, 3), (3, 1), (2, 1)\rangle$ (due to a single left shift) and $\langle(1, 3), (3, 1), (1, 2)\rangle$ (due to an inversion). Consequently, two subsolutions S_i and S_j can be spliced together in $2 \times |S_i| \times |S_j|$ different ways. This gives rise to the following definition.

Definition 4. Let S_i and S_j be two subsolutions. A *minimum cost splice* is the operation of splicing S_i and S_j such that the minimum additional cost, denoted by $\rho(S_i, S_j)$, is incurred.

The value for $\rho(S_i, S_j)$ is calculated by simply checking all $2 \times |S_i| \times |S_j|$ possible splicing options and taking the smallest value for Eq. (4) among these. Splicing these subsolutions then involves rotating and inverting S_i and S_j appropriately, before appending one to the other. As an example, consider the subsolutions $S_i = \langle(2, 1), (1, 3), (3, 1)\rangle$ and $S_j = \langle(5, 4), (4, 5)\rangle$. Here, the minimum cost splice is achieved by left-shifting S_i by two positions, left-shifting S_j by one position, and then appending to form the new subsolution $\langle(3, 1), (2, 1), (1, 3), (4, 5), (5, 4)\rangle$. Using Eq. (4), this splice brings the (minimal) additional cost of $\rho(S_i, S_j) = f(3, 4) + f(4, 3) - f(3, 3) - f(4, 4) = 1 + 1 - 0 - 0 = 2$.

In the (unproven) version of EULER-SPLICE proposed by Lewis and Holborn (2017), the l subsolutions are spliced into a single (optimal) solution using an $\mathcal{O}(n^3)$ -time process. At each iteration of this process a value $\rho(S_i, S_j)$ is calculated for each pair of subsolutions S_i, S_j , which takes $\mathcal{O}(n^2)$ time. The splice corresponding to the minimal observed value for $\rho(S_i, S_j)$ is then performed, reducing the number of subsolutions by one, and the algorithm then repeats. In the worst case, this process starts with $l = n$ different subsolutions, resulting in n separate applications of this $\mathcal{O}(n^2)$ process, resulting in the noted cubic complexity. Here, we propose the more efficient SPLICE method given by Algorithm 2. As shown, this operates by first forming a minimum spanning tree T in which each vertex v_i represents a subsolution S_i , and the weight of each edge $\{v_i, v_j\}$ in T is $\rho(S_i, S_j)$. The tree T now describes how the various subsolutions should be spliced together to form a single solution. Moreover, under the cost function $f(x, y) = \|x - y\|$, this solution will be optimal, and the additional costs incurred by performing the splices will equal the total of the edge weights in T (see Theorem 4).

We now consider the overall complexity of this modified EULER-SPLICE algorithm for a given problem instance P containing n elements.

Algorithm 2: SPLICE (quadratic-time method)

input : A set of sub-solutions $\{S_1, S_2, \dots, S_l\}$.
output : A single solution.

- 1 Let $G = (V, E)$ be a complete, undirected edge-weighted graph with vertex set $V = \{v_1, \dots, v_l\}$ and edge weights $w(v_i, v_j) = \rho(S_i, S_j)$, and let $T = (V, E_T)$ be a minimum spanning tree on G .
- 2 **while** the number of vertices in T is greater than one **do**
- 3 Let v_i be a leaf vertex in T and let v_j be its (only) neighbour.
- 4 Splice S_i into S_j (incurring an additional cost of $\rho(S_i, S_j)$) and remove v_i from T .
- 5 **return** S_j

First, the construction of the weighted multigraph (Definition 3) takes $\mathcal{O}(|V| \lg |V|)$ time because vertices are ordered by weight. In the worst case, all vertices will feature a degree of one, making this operation $\mathcal{O}(n \lg n)$. Adding the dummy edges is then carried out in $\mathcal{O}(n)$ time. The actions of identifying the Eulerian components (via Hierholzer’s algorithm) and converting each C_i to a corresponding solution S_i are also linear in complexity. When splicing the subsolutions using Algorithm 2, the construction of the complete edge-weighted graph is an $\mathcal{O}(n^2)$ process, while the calculation of T takes $\mathcal{O}(l^2)$ time (which is also $\mathcal{O}(n^2)$ in the worst case). Finally, the splicing on Line 4 of SPLICE can be performed in constant time providing linked lists (or similar) are used. The overall worst-case complexity of this version of EULER-SPLICE is therefore $\mathcal{O}(n^2)$.

3.2. EULER-SPLICE adaptation for the OPSP

To adapt EULER-SPLICE to solve the OPSP (Definition 2) in $\mathcal{O}(n^2)$ time, three modifications are necessary. First, Definition 3 should be modified so that an arc-weighted directed multigraph $G = (V, A)$ is produced for which $A = P$. All other characteristics of the graph remain the same. Second, the matching process used to make the vertex degrees of G even (Lines 2–6 of Algorithm 1) should now be replaced by Algorithm 3. This new $\mathcal{O}(n)$ process adds dummy arcs to

Algorithm 3: Matching Procedure for the OPSP (replaces Lines 2–6 of Algorithm 1)

input : Directed graph $G = (V, A)$.
output : Updated version of G that is Eulerian.

- 1 Set L_{in} and L_{out} as empty lists.
- 2 **foreach** $i \in \{1, \dots, |V|\}$ **do**
- 3 $d = \deg^+(v_i) - \deg^-(v_i)$
- 4 **if** $d < 0$ **then foreach** $j \in \{1, \dots, -d\}$ **do** append v_i to L_{in}
- 5 **else if** $d > 0$ **then foreach** $j \in \{1, \dots, d\}$ **do** append v_i to L_{out}
- 6 **foreach** $i \in \{1, \dots, |L_{\text{in}}|\}$ **do**
- 7 Let $u = L_{\text{out}}[i]$ and let $v = L_{\text{in}}[i]$
- 8 Add a dummy arc (u, v) to G and set $w(u, v) = f(w(u), w(v))$

G so that the in-degree and out-degree of each vertex are made equal (i.e., $\deg^+(v) = \deg^-(v) \forall v \in V$), ensuring that the resultant directed components C_1, C_2, \dots, C_l are Eulerian. Finally, when calculating values for $\rho(S_i, S_j)$ (Algorithm 2, Line 1), inversions should not be considered. Consequently, only $|S_i| \times |S_j|$ splicing options are evaluated for each pair of subsolutions S_i, S_j .

A worked example of this modified algorithm is shown in Fig. 3(b). Note that an alternative way of solving the OPSP is to use the noted $\mathcal{O}(n^2)$ algorithm of Gilmore and Gomory (1964), setting the functions in Eq. (1) as $g_1(z) = g_2(z) = 1$.

3.3. Proof of correctness

In this section, we prove the correctness of our $\mathcal{O}(n^2)$ EULER-SPLICE algorithm for both the PSP and the OPSP. As we have seen, the first part of the algorithm involves generating the graph $G = (V, E)$ from Definition 3 and then, if necessary, adding a minimum-weight set of dummy edges/arcs between pairs of odd-degree vertices to make G Eulerian. A suitable approach for generating this set is to use the blossom algorithm to produce a minimum-cost perfect matching between the odd-degree vertices.¹ Here, however, the particular structure of G allows the set of dummy edges/arcs to be determined using the less expensive linear time processes given in Algorithms 1 and 3 as we now show.

Theorem 2. In EULER-SPLICE, the sum of the weights of the dummy edges added to G is minimal for the PSP. This is also true for the sum of the weights of the dummy arcs added to G with the OPSP.

Proof. Consider the PSP first. Given the graph $G = (V, E)$ from Definition 3, let $G' = (V', E')$ be a complete edge-weighted graph comprising the odd-degree vertices of G and edge weights $f(w(u), w(v))$ for each $u, v \in V'$. Note that, according to the handshaking lemma, $|V'|$ will be even. Now imagine that each vertex $v \in V'$ is placed on the number line at position $w(v)$. Since this is a straight line, a minimum-cost perfect matching can be determined by simply matching each successive pair of vertices on this line from left to right. It can be seen that the set of weighted edges that are added to G in Lines 2–6 of Algorithm 1 is equivalent to this matching. These additional edges ensure that the degrees of all vertices are even, making G Eulerian, as required.

For the OPSP, given the directed graph $G = (V, A)$, let $G' = (V', A')$ be an arc-weighted graph for which the vertex set is partitioned into a set of black vertices and a set of red vertices. Specifically, for each

$v \in V$, if $\deg^-(v) > \deg^+(v)$ then $\deg^-(v) - \deg^+(v)$ black copies of v (with weight $w(v)$) are added to V' ; else, if $\deg^+(v) > \deg^-(v)$ then $\deg^+(v) - \deg^-(v)$ red copies of v (with weight $w(v)$) are added to V' . Finally, for each pair of vertices $u, v \in V'$ such that u is red and v is black, an arc (u, v) of weight $f(w(u), w(v))$ is added to A' . Similar reasoning to the previous case can now be applied. Due to the handshaking lemma for directed graphs, the number of black vertices will equal the number of red vertices, making $|V'|$ even. We also place the vertices of V' onto the number line at the positions given by their weights. A minimum-cost perfect matching can now be determined using the following process. Moving from left to right, if an unmatched red vertex u is encountered, then this is matched to the nearest black vertex v to the right of u using the arc (u, v) . Similarly, if an unmatched black vertex v is encountered, then this is matched to the nearest red vertex u to the right of v using the arc (u, v) . The set of weighted arcs that are added to G in Algorithm 3 is equivalent to this matching and, as required, the in-degree of each vertex in G will now equal its out-degree, making G Eulerian. \square

At Line 11 of EULER-SPLICE, each subsolution S_1, \dots, S_l maps, respectively, to the Eulerian component C_1, \dots, C_l in G . The cost of each subsolution S_i is also equal to the sum of the weights of the dummy edges (arcs) present in C_i . The total cost of the l subsolutions, $\sum_{i=1}^l \mathcal{F}(S_i)$, is therefore equal to the sum of the weights of all dummy edges (arcs) in G , implying that the former is also minimal.

Theorem 3. For both the PSP and OPSP, if, at Line 11 of EULER-SPLICE, just $l = 1$ subsolution has been produced, then this is a full optimal solution. Furthermore, the cost of this solution will be zero if and only if no dummy edges were added to G in the preceding steps.

Proof. Consider the PSP. If $l = 1$ then $C_1 = G$; hence G is a connected Eulerian graph. According to Definition 3, all edges in this graph have a weight of zero, except for the dummy edges. Since the sum of the weights of these dummy edges is minimal, the cost of the solution S_1 is also minimal.

If the above is true and, in addition, no dummy edges were added to G , then the sum of the edge weights in C_1 will be zero. Conversely, consider the (full) solution S_1 that has a cost of zero. W.l.o.g. this can be written as a sequence of ordered pairs $S_1 = \langle (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n) \rangle$ in which $y_i = x_{i+1} \forall i \in \{1, \dots, n-1\}$, and $y_n = x_1$. Seen as a series of edges, this gives an Eulerian circuit.

For the OPSP we apply the same arguments and simply replace the term “edge” with “arc” in the above. \square

At Line 11 of EULER-SPLICE, in cases where $l \geq 2$ subsolutions have been formed, it is necessary to merge these into a single, full solution using the SPLICE method (Algorithm 2). As shown, this algorithm starts with the production of a new complete, undirected, edge-weighted graph $G = (V, E)$ in which each vertex $v_i \in V$ corresponds to the subsolution S_i , and edge weights $w(v_i, v_j) = \rho(S_i, S_j)$. By definition, a minimum spanning tree $T = (V, E_T)$ is a spanning subgraph of G whose edge-weight sum is minimal among all such subgraphs (Cormen et al., 2009). Consequently, it is sufficient to show that, by executing Lines 2–4 of Algorithm 2, the act of splicing a (leaf) subsolution S_i into its neighbour S_j will not change the values of the minimum cost splices between any remaining pairs of subsolutions (including S_j). If this is the case, then T specifies a way of splicing the subsolutions S_1, \dots, S_l into a single solution whose cost

$$\sum_{i=1}^l \mathcal{F}(S_i) + \sum_{\{v_i, v_j\} \in E_T} w(v_i, v_j) \quad (5)$$

is minimal.

Theorem 4. Given the set of subsolutions $\{S_1, \dots, S_l\}$ produced at Line 11 of EULER-SPLICE and using $f(x, y) = \|x - y\|$, the SPLICE method (Algorithm 2) gives a single optimal solution for both the PSP and OPSP.

¹ Several variants of the blossom algorithm are noted in the literature, featuring complexities such as $\mathcal{O}(|E||V|^2)$, $\mathcal{O}(|V|^3)$, and $\mathcal{O}(|E||V| \lg |V|)$. A survey on this matter is presented by Duan and Pettie (2014).

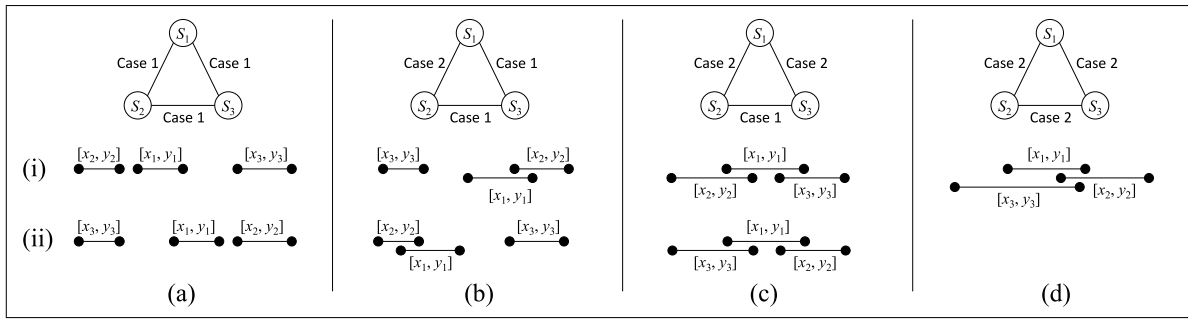


Fig. 4. The four possible ways of merging subsolutions of the OPSP using EULER-SPLICE.

Proof. Consider the OPSP first. Note that $\forall \{v_i, v_j\} \in E_T, w(v_i, v_j) \geq 0$, as to be otherwise would imply that the cost of the matching from the previous step was not minimal. W.l.o.g., now assume that T features the edges $\{v_1, v_2\}$ and $\{v_1, v_3\}$, and that v_2 is a leaf vertex. This implies that $\rho(S_1, S_2) \leq \rho(S_2, S_3)$ and that $\rho(S_1, S_3) \leq \rho(S_2, S_3)$. According to the behaviour of SPLICE, we will now combine S_1 and S_2 to form the subsolution $S_{\{1,2\}}$, incurring an additional cost of $\rho(S_1, S_2)$. It is now sufficient to show that

$$\rho(S_1, S_3) = \rho(S_{\{1,2\}}, S_3) \quad (6)$$

in all cases.

Consider first the act of splicing S_1 and S_2 . For convenience, assume that these subsolutions are already appropriately left-rotated, allowing a minimum cost splice to be performed by appending one to the other. These subsolutions can be written as $S_1 = \langle (x_1 \dots y_1) \rangle$ and $S_2 = \langle (x_2 \dots y_2) \rangle$, with

$$\rho(S_1, S_2) = f(y_1, x_2) + f(y_2, x_1) - f(x_1, y_1) - f(x_2, y_2). \quad (7)$$

Splicing S_1 and S_2 therefore leads to the solution $S_{\{1,2\}} = \langle (x_1, \dots, y_1), (x_2, \dots, y_2) \rangle$.

Now consider the minimal cost splice for S_1 and S_3 . If this involves inserting S_3 between two internal elements of S_1 —specifically, not between y_1 and x_1 —then it is obvious that Eq. (6) holds. Consequently, we now only need to consider the situation where the minimal cost splice for S_1 and S_3 is achieved by inserting S_3 between y_1 and x_1 . If this is the case, then it is necessary that

$$\rho(S_1, S_3) = \rho(S_{\{1,2\}}, S_3) = \min(f(y_1, x_3) + f(y_3, x_2) - f(y_1, x_2) - f(x_3, y_3), f(y_2, x_3) + f(y_3, x_1) - f(y_2, x_1) - f(x_3, y_3)). \quad (8)$$

To complete the proof, we now need to consider the characteristics of the chosen function $f(x, y) = \|x - y\|$. Let $S_i = \langle (x_i \dots y_i) \rangle$ and $S_j = \langle (x_j \dots y_j) \rangle$ be two subsolutions, written as specified above. Because f is commutative, we can assume that $x_i \leq y_i$. Similarly, because Eq. (7) is commutative, we can assume $x_j \leq y_j$. In splicing S_i and S_j to form $S_{\{i,j\}} = \langle (x_i, \dots, y_i), (x_j, \dots, y_j) \rangle$ there are now four possibilities to consider.

- (i) $x_i \leq y_i \leq x_j \leq y_j$. Here, Eq. (7) simplifies to $2 \cdot f(y_i, x_j)$. If $y_i \neq x_j$ this equates to a positive value, else it gives zero.
- (ii) $x_i \leq y_i \leq y_j < x_j$. Similarly, in this case Eq. (7) simplifies to $2 \cdot f(y_i, y_j)$ giving a positive value if $y_i \neq y_j$ and zero otherwise.
- (iii) $x_i \leq x_j < y_i \leq y_j$. Here, Eq. (7) equates to zero, since $f(x_i, y_i) + f(x_j, y_j) = f(x_i, y_j) + f(x_j, y_i)$.
- (iv) $x_i \leq x_j \leq y_j < y_i$. Similarly, in this case Eq. (7) equates to zero, since $f(x_i, y_i) + f(x_j, y_j) = f(x_i, y_i) + f(x_j, y_i)$.

(The remaining two possibilities, namely $x_i \leq y_j < y_i \leq x_j$ and $x_i \leq y_j < x_j < y_i$, cannot occur as they equate to negative values.) These four possibilities can be combined into two cases:

Case 1: If the real intervals defined by $[x_i, y_i]$ and $[x_j, y_j]$ do not intersect, then $\rho(S_i, S_j) = 2 \cdot d([x_i, y_i], [x_j, y_j])$, where

$$d([x_i, y_i], [x_j, y_j]) = \min(\|x_i - x_j\|, \|x_i - y_j\|, \|y_i - x_j\|, \|y_i - y_j\|) \quad (9)$$

gives the distance between the two closest endpoints belonging to the different intervals. In this case, $\rho(S_i, S_j) > 0$.

Case 2: Otherwise, the intervals $[x_i, y_i]$ and $[x_j, y_j]$ intersect (or are adjacent), giving $\rho(S_i, S_j) = 0$.

Now consider the splicing of the three subsolutions S_1, S_2, S_3 , as mentioned earlier. Splicing S_1 and S_2 produces the subsolution $S_{\{1,2\}} = \langle (x_1, \dots, y_1), (x_2, \dots, y_2) \rangle$ and, in effect, replaces the intervals $[x_1, y_1]$ and $[x_2, y_2]$ with the intervals $[x_1, y_2]$ and $[x_2, y_1]$. Four options are now possible, which are illustrated in Fig. 4(a)–(d). For Fig. 4(a) and (b), we have

$$\rho(S_{\{1,2\}}, S_3) = \min(2 \cdot d([x_1, y_2], [x_3, y_3]), 2 \cdot d([x_2, y_1], [x_3, y_3])) = \rho(S_1, S_3) \quad (10)$$

as required. In Fig. 4(c) and (d), meanwhile, we see that $[x_1, y_1]$ and $[x_3, y_3]$ intersect, giving $\rho(S_1, S_3) = 0$. By inspection, it can be seen that at least one of the intersections $[x_1, y_2]$ and $[x_2, y_1]$ also intersects with $[x_3, y_3]$. Hence $\rho(S_{\{1,2\}}, S_3) = \rho(S_1, S_3) = 0$.

Finally, for the PSP, observe that for Case 2 above, the allowance of left-rotations will result in negative values for $\rho(S_i, S_j)$; however, these negative values cannot occur during execution of the overall EULER-SPLICE method because they would imply that the matching used in previous steps was not minimal. As a result, instances of Case 2 cannot occur with the PSP. All other details remain the same. \square

4. A generalised approach

The previous section gave an exact $\mathcal{O}(n^2)$ algorithm for the PSP and OPSP under the function $f(x, y) = \|x - y\|$. Other functions can also be encountered in bar nesting problems, however. One example is illustrated in Fig. 5 where, unlike earlier, the trapezoidal items cannot be up-rotated. In industrial applications, this situation occurs when the cross sections of the stocks have no symmetry (such as the L-shaped girder shown in Fig. 1(c)), or when different sides of the stocks have different properties (such as non-slip surfaces on timber decking). In this particular case, it is now necessary to use the function $f(x, y) = \|x + y\|$, where negative values are used for projections occurring on the top of an item, and positive values are used for those at the bottom (as shown in Fig. 5).

As we saw earlier, the use of the function $f(x, y) = \|x - y\|$ allows the PSP to be reformulated using the graph from Definition 3, where each vertex can be seen as occurring on a number line, and each edge weight corresponds to the Euclidean distance between its endpoints. Under other functions, such as $\|x + y\|$, this is not the case so a more general formulation and solution method is required.

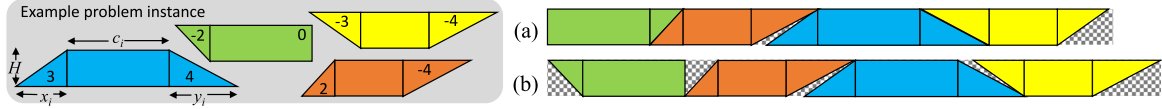


Fig. 5. A small trapezoid packing problem instance involving four items. Part (a) shows an optimal arrangement when only left rotations are permitted; Part (b) shows the optimum solution when no rotations are permitted.

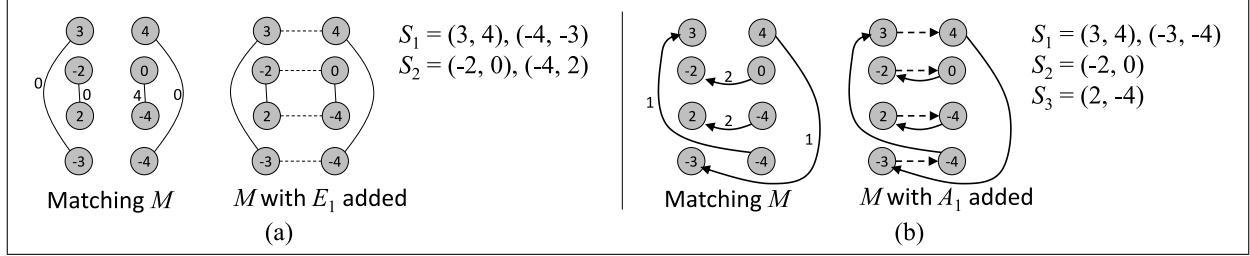


Fig. 6. Demonstration of the MATCH-SPLICE method for (a) the GPSP, and (b) and GOPSP, using the problem instance from Fig. 5 (with $f(x, y) = \|x + y\|$).

Definition 5. The Generalised Pair Sequencing Problem (GPSP) is equivalent to the PSP (Definition 1), except that $f(x, y)$ is now an arbitrary function. Similarly, the Generalised Ordered Pair Sequencing Problem (GOPSP) is equivalent to the OPSP (Definition 2) under an arbitrary function $f(x, y)$.

Given a problem instance $P = \{\{x_1, y_1\}, \dots, \{x_n, y_n\}\}$, the GPSP can be modelled by a complete, edge-weighted graph $G = (V, E)$. This is done by considering each of the $2n$ elements in P as a vertex, adding edge weights of $-\infty$ between vertices belonging to the same pair $\{x_i, y_i\} \in P$, and edge weights of $f(x, y)$ between all remaining pairs. A minimum-cost Hamiltonian cycle in this graph then corresponds to a minimum-cost solution to the GPSP.² Clearly, the task of identifying such a cycle is equivalent to the TSP and, under arbitrary functions, will be \mathcal{NP} -hard. That said, several functions encountered in industrial bar nesting problems lead to polynomially solvable special cases of the TSP. In this section, we will refer to such functions as “admissible” and present several examples.

4.1. An exact solution method and admissible functions

Recall that, in Lines 7–10 of EULER-SPLICE, each connected component $C_1 \dots, C_l$ is transformed into a corresponding subsolution $S_1 \dots, S_l$ such that the total cost of the subsolutions $\sum_{i=1}^l \mathcal{F}(S_i)$ is minimal. The same set of connected components can be formed following the steps of the MATCH-SPLICE method given in Algorithm 4. As shown, this involves constructing a complete graph with $2n$ vertices in which edge weights are defined by an admissible function $f(x, y)$. A minimum-cost perfect matching M is then determined in this graph, and these edges are combined with the set E_1 to form $C_1 \dots, C_l$. These are then spliced into a single solution as before. Worked examples of this algorithm are given in Fig. 6. Note that, in this case, the blossom algorithm is used for determining the matching M ; consequently, the overall complexity of MATCH-SPLICE is dominated by the complexity of the chosen blossom variant (see Section 3.3) and is therefore higher than $\mathcal{O}(n^2)$.

The question to now ask is “which functions are admissible?” That is, which functions allow the SPLICE method (Algorithm 2) to produce a single solution that guarantees the satisfaction of Eq. (5), therefore giving an optimal solution? Theorem 4 previously demonstrated the admissibility of $\|x - y\|$. However, note that the function itself was not considered in the proof until Eq. (8). To prove a function’s admissibility it is therefore sufficient to simply demonstrate the truth of Eq. (8) with respect to this function.

² The same statements can be made for the GOPSP. In this case, we simply use arcs instead of edges in the appropriate manner.

Algorithm 4: MATCH-SPLICE method for the GPSP.

input : A problem instance P and admissible function $f(x, y)$.
output : A single optimal solution.


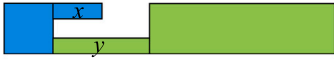
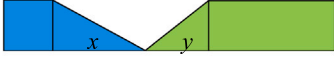


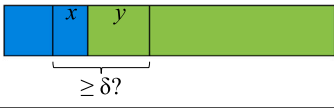
- 1 Let $V = \emptyset$, $E_1 = \emptyset$ and $E_2 = \emptyset$.
- 2 **foreach** $\{x_i, y_i\} \in P$ **do**
- 3 Add vertices v_{2i-1} and v_{2i} to V , and set their weights
 $w(v_{2i-1}) = x_i$ and $w(v_{2i}) = y_i$.
- 4 Add the edge $\{v_{2i-1}, v_{2i}\}$ to E_1 , and set its weight
 $w(v_{2i-1}, v_{2i}) = f(x_i, y_i)$.
- 5 **foreach** pair of vertices $u, v \in V : \{u, v\} \notin E_1$ **do**
- 6 Add the edge $\{u, v\}$ to E_2 and set its weight
 $w(u, v) = f(w(u), w(v))$.
- 7 Use the blossom algorithm to produce a minimum-cost perfect matching M on the graph $(V, E_1 \cup E_2)$.
- 8 Let C_1, \dots, C_l be the connected components (cycles) of the multigraph $G = (V, E_1 \uplus M)$. Now apply Lines 8–12 of EULER-SPLICE (Algorithm 1), using M as the set of dummy edges.

Table 1 lists several admissible functions with descriptions of how they are directly relevant in bar nesting problems. The first three examples in the table are proven in this paper. The remaining three are conjectured, though we have strong empirical evidence of their admissibility: first, we have executed MATCH-SPLICE with these functions on large numbers of problem instances and have always found Eq. (5) to be satisfied; second, we have made considerable use of the Python script listed in Appendix B to test the admissibility of these functions. This Python script is also useful for quickly identifying inadmissible functions. Examples of the latter include $\min(x, y)$ (even though $\max(x, y)$ is admissible), $x \times y$, the Kronecker delta function and, naturally, when $f(x, y)$ maps to randomly selected values. Note that only the first function in Table 1 can be expressed in the form given by Eqs. (1) and (2). The method of Gilmore and Gomory (1964) is therefore unsuitable for the remaining cases.

5. Empirical analysis

In this section, we provide information on the performance of our algorithms using differing problem instances and functions. Here, algorithms were implemented in C++ and executed on Windows 10 machines with 3.5 GHz quad-core CPUs and 8 GB RAM. For MATCH-SPLICE, the highly-regarded implementation of Kolmogorov (2009) was

Table 1
Admissible functions with illustrated examples. Problem instances defined by these functions are solved by the MATCH-SPLICE method.

$f(x, y)$ with illustration	Notes
$\ x - y\ $ 	(Theorem 4.) Under this function, problem instances are solved by both EULER-SPLICE and MATCH-SPLICE, though the former has a lower complexity.
$\max(x, y)$ 	(Theorem 5, Appendix B.) Models the situation where wastage between two shapes is defined by the maximum of the sizes of the two adjacent projections, as illustrated.
$\ x\ + \ y\ $ 	(Theorem 6, Appendix B.) Models the situation where, for example, items cannot be up-rotated, and all projections occur on the same side of the items.
$\ x + y\ $ 	Models the situation where items cannot be up-rotated, but projections can occur on either side of the items. (See also Figure 5.)
$\ x - y\ $ or $\ x + y\ $ 	Models the situation where consecutive cuts can be made on different planes. For example, the figure considers a stock with a rectangular cross-section. Here, the adjacent projections (of size x and y) have been produced by rotating the stock by 90° between the two cuts, meaning that the projections occur on different planes. When adjacent projections occur on different planes, wastage is calculated as $\ x + y\ $, else it is calculated as $\ x - y\ $.
$\delta - x - y$ if $x + y < \delta$, 0 otherwise 	Here, a zero-cost solution corresponds to a feasible solution of the minimum score separation problem (Section 2). The use of this function with MATCH-SPLICE therefore offers an alternative to the $\mathcal{O}(n^2)$ method of Hawa et al. (2022) and also solves the problem when left-rotations of items are forbidden.

used for the blossom algorithm. Problem instances, meanwhile, were generated by randomly selecting values for each x_i and y_i ($i \in \{1, \dots, n\}$) from a user-defined integer range. Our C++ code, problem instance generator, a full listing of results, and a Python implementation of these algorithms are available online (Lewis, 2024).

Fig. 7 demonstrates the time requirements of our $\mathcal{O}(n^2)$ EULER-SPLICE method. In general, the algorithm proves to be very fast, with problems of up to $n = 20000$ elements being solved in less than 0.1 s. Execution times also fall slightly when problem instances are generated using smaller integer ranges. This is because, in these cases, minimum spanning trees with fewer vertices tend to be formed during the execution of the SPLICE algorithm; hence this part of the algorithm completes more quickly. EULER-SPLICE is also marginally faster with the OPSP, because fewer options need to be considered for the calculation of each $\rho(S_i, S_j)$.

Fig. 8 shows results from similar experiments with MATCH-SPLICE under three admissible functions. For experimental purposes, the third function used here considers two projections to be on the same plane (therefore applying $\|x - y\|$) only when x and y are either both odd or both even; otherwise $\|x + y\|$ is used. Compared to EULER-SPLICE, the runtimes of this algorithm are longer and are also influenced by the chosen function and integer ranges. In all of these trials, we found that less than 10% of the execution time was used by the splicing part of the algorithm, indicating that the blossom algorithm is by far the most costly part of the overall procedure.

Fig. 9 shows the execution times of MATCH-SPLICE when using the function that solves the minimum score separation problem. As before, we see that run times increase for larger problem instances; however, we also see that run times are short when δ is relatively small or large. In these cases, the blossom algorithm completes quickly, usually determining a zero-cost matching for the former and a positive-cost

matching for the latter. On the other hand, intermediate values of δ lead to matching problems that take the blossom algorithm much longer to solve.

Finally, results from our last set of experiments are shown in Fig. 10. Here, we use the arbitrary inadmissible function $f(x, y) = \|x \times y\| \pmod{101}$ to compare MATCH-SPLICE to a well-known $\mathcal{O}(|E| \lg |E|)$ greedy heuristic for the TSP.³ As noted, the use of an inadmissible function here means that Eq. (5) is not always satisfied and, as a result, MATCH-SPLICE is only approximative. It also means that the splicing part of the algorithm now has the higher complexity of $\mathcal{O}(n^3)$, as was described in Section 3.1. The results indicate that, while the run times are longer for MATCH-SPLICE, particularly for large problem instances, the costs of its solutions are markedly lower. MATCH-SPLICE may therefore offer a promising, albeit rather expensive, heuristic for the GPSP and OGPSP under inadmissible functions.

6. Conclusions

In this paper, we have presented polynomial-time methods that solve several single-stock bar nesting problems arising in industry. Because these represent special cases of the TSP, as a byproduct we have also identified new members in the set of TSP cases that are polynomially solvable. (Other members of this set are surveyed by Burkard et al., 1998.) For multi-stock variants of these problems, our methods could

³ This heuristic operates by considering each edge of the graph in ascending weight order and adding edges to the solution so that no vertex in the solution has a degree of more than two, and so the solution only contains a cycle once the n th edge is added. As a variant of Kruskal's algorithm, its complexity is $\mathcal{O}(|E| \lg |E|)$ (Cormen et al., 2009; Khemani, 2008).

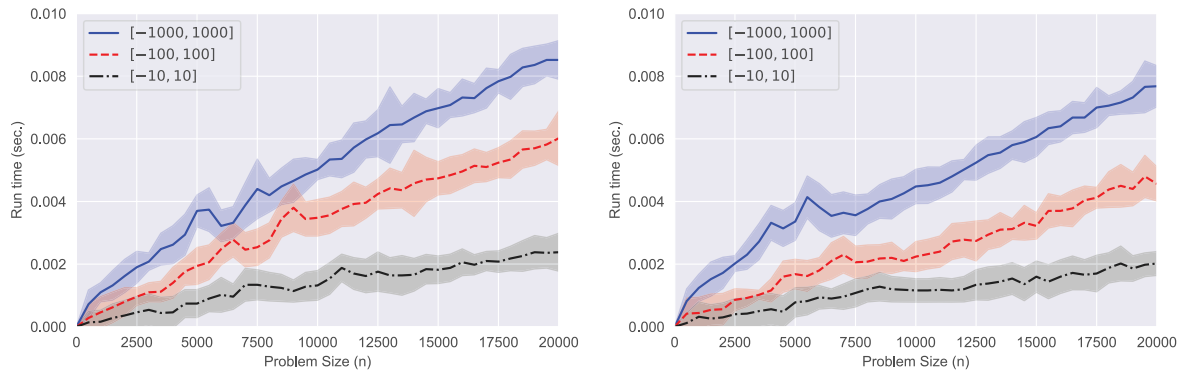


Fig. 7. Mean run times of the EULER-SPLICE algorithm for the PSP (left) and OPSP (right) for differing problem sizes and integer ranges. Shaded areas indicate one standard deviation on either side of the mean. Values for each n and range are calculated across fifty randomly generated problem instances.

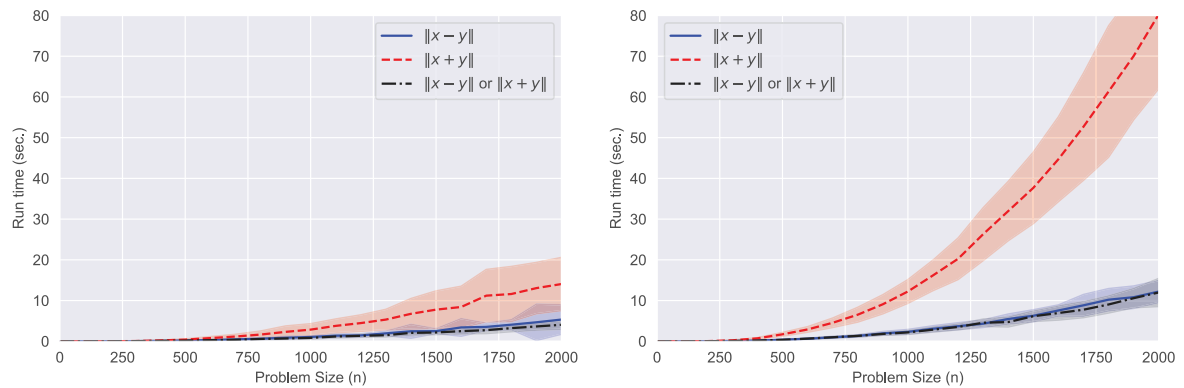


Fig. 8. Mean run times of the MATCH-SPLICE algorithm for the GPSP using problem instances of ranges $[-10, 10]$ (left) and $[-1000, 1000]$ (right) under different admissible functions. Remaining details are the same as Fig. 7.

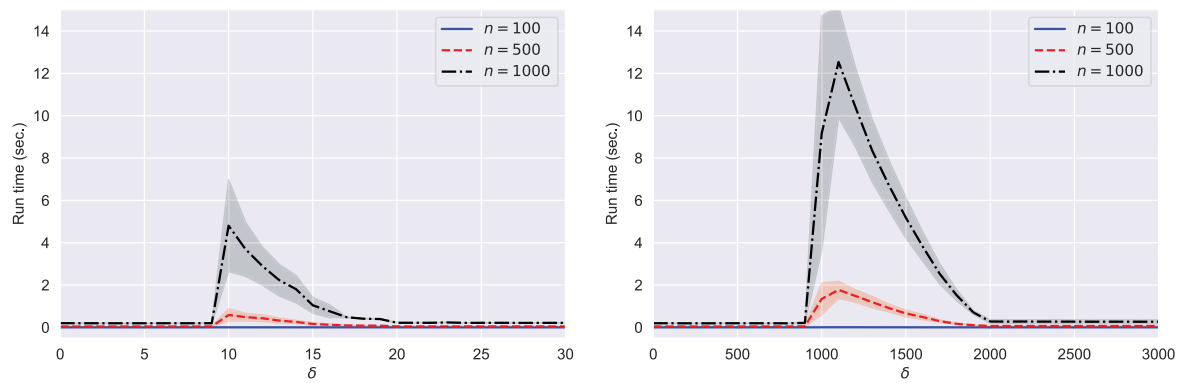


Fig. 9. Mean run times of the MATCH-SPLICE algorithm for the GPSP using problem instances of ranges $[0, 10]$ (left) and $[0, 1000]$ (right) with $f(x, y) = \delta - x - y$ if $x + y < \delta$ and 0 otherwise, and varying values for δ . Remaining details are the same as Fig. 7.

be used as sub-procedures in a wider algorithmic framework, which could include techniques such as constructive heuristics, local search, and/or column generation. In such frameworks, our exact algorithms may need to be executed many times; however, they will also be able to halt early in many cases because the employed matching process gives a lower bound on inter-item wastage. Consequently, if this lower bound exceeds the permitted wastage, then it is known the items of concern cannot be cut from a single stock.

As mentioned in the introduction, this paper has focussed on applications where the aim is to cut a predefined set of items from a minimal number of stocks. In industrial settings, it is common for each item i to be associated with a given demand $d_i \in \mathbb{N}$, specifying that

d_i copies of item i are required. Alternative problem formulations also exist in which the number of stocks is limited, and the objective is to identify the best subset of items that can be cut from these stocks so that demands are not exceeded. Wäscher et al. (2007) calls this formulation a “fixed-dimension output-maximisation problem” and surveys several algorithms that could be combined with our suggested methods for this multi-stock variant.

Our exchanges with TopSolid have indicated that it is rare to see more than around 15 or 20 items per stock. This suggests that the scaling-up features of our algorithms are unlikely to be problematic in industrial settings. That said, it would be interesting in future work to see whether the matching problems for the admissible functions

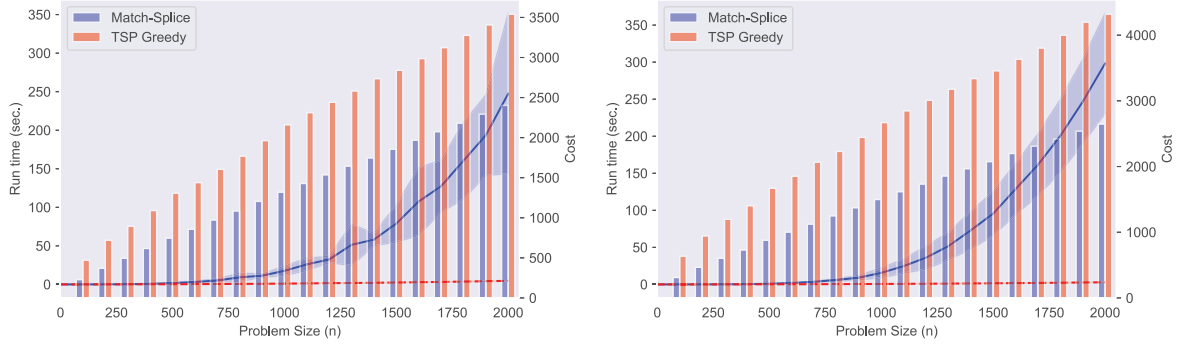


Fig. 10. Here, bars compare the costs of solutions produced by MATCH-SPLICE and the greedy TSP heuristic for instances of the GPSP (left) and GOPSP (right) using an inadmissible function. Lines show the corresponding run times of the algorithms, as with previous figures.

listed in Table 1 can be solved by less expensive processes, as we have achieved with the function $\|x - y\|$ in Algorithms 1 and 3.

CRedit authorship contribution statement

Rhyd Lewis: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Louis Bonnet:** Writing – review & editing, Visualization, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Additional proofs

Theorem 5. *The function $f(x, y) = \max(x, y)$ is admissible.*

Proof. The initial steps of this proof are identical to those of Theorem 4. They diverge at the point where the four possibilities are considered. For $f(x, y) = \max(x, y)$, these are as follows.

- (i) $x_i \leq y_i \leq x_j \leq y_j$. Here, evaluating each occurrence of f in Eq. (7) gives $y_j + x_j - y_i - y_j = x_j - y_i$.
- (ii) $x_i \leq y_i \leq y_j < x_j$. Similarly, in this case applying Eq. (7) gives $y_j + x_j - y_i - x_j = y_j - y_i$.
- (iii) $x_i \leq x_j < y_i \leq y_j$. Here, Eq. (7) equates to zero, because the evaluation of each occurrence of f leads to $y_j + y_i - y_i - y_j = 0$.
- (iv) $x_i \leq x_j \leq y_j < y_i$. Similarly, in this case, we get $y_j + y_i - y_i - y_j = 0$.

As before, the remaining two possibilities cannot occur as they would equate to negative values. These four possibilities can now be combined into two cases:

Case 1: If the intervals $[x_i, y_i]$ and $[x_j, y_j]$ do not intersect, then $\rho(S_i, S_j) = d([x_i, y_i], [x_j, y_j]) > 0$.

Case 2: Otherwise, $[x_i, y_i]$ and $[x_j, y_j]$ intersect (or are adjacent), giving $\rho(S_i, S_j) = 0$.

The remaining steps of the proof are identical to Theorem 4, except that Eq. (10) is replaced by

$$\rho(S_{\{1,2\}}, S_3) = \min(d([x_1, y_2], [x_3, y_3]), d([x_2, y_1], [x_3, y_3])) = \rho(S_1, S_3). \quad (11)$$

Theorem 6. *The function $f(x, y) = \|x\| + \|y\|$ is admissible.*

Proof. By definition, $\|x\|$ and $\|y\|$ are nonnegative so, for any values x_i, y_i, x_j, y_j , we get:

$$\begin{aligned} & f(x_i, y_j) + f(x_j, y_i) - f(x_i + y_i) - f(x_j, y_j) \\ &= x_i + y_j + x_j + y_i - x_i - y_i - x_j - y_j \\ &= 0. \end{aligned} \quad (12)$$

Hence, in all cases, $\rho(S_{\{1,2\}}, S_3) = \rho(S_1, S_3) = 0$. \square

Appendix B. Empirical verification of function admissibility

The following Python program uses a series of random trials to test whether a particular function $f(x, y)$ is admissible. The function is specified at Line 4 and values for other parameters are given at Line 12. On execution, if Line 31 is reached, this gives evidence that f is admissible; otherwise, an example is printed that proves that f is inadmissible. The use of the matching M at Line 24 ensures that x_1, y_1, x_2, y_2, x_3 , and y_3 have been assigned values that are permitted to occur in applications of SPLICE.

The implementation of getMatching() shown here concerns the GOPSP. For the GPSP, additional edges between each pair of x values and each pair of y values are required.

```

1 import random, sys, networkx as nx
2
3 def f(x, y):
4     return abs(x-y)
5
6 def getMatching():
7     G = nx.Graph()
8     G.add_weighted_edges_from([("x1", "y1", f(x1, y1)), ("x2", "y2", f(x2, y2)), ("x3",
9         "y3", f(x3, y3)), ("y1", "x2", f(y1, x2)), ("y1", "x3", f(y1, x3)), ("y2", "x1",
10         f(y2, x1)), ("y2", "x3", f(y2, x3)), ("y3", "x1", f(y3, x1)), ("y3", "x2", f(y3,
11         x2))])
12     M = nx.min_weight_matching(G,
13         maxcardinality=True, weight="weight")
14     return sorted([sorted(list(edge)) for
15         edge in M])
16
17 LB, UB, numTrials, cnt = -10, 10, 5000, 0
18 for i in range(numTrials):
19     #Randomly select values for the
20     #variables in the given range and
21     #find the min-cost perfect matching M
22     #of the associated graph
23     x1, x2, x3, y1, y2, y3 = [random.randint
24         (LB, UB) for i in range(6)]

```

```

16 M = getMatching()
17 #Calculate the cost of each splice and
   store them in the dictionary rho
18 rho = dict()
19 rho["S1", "S2"] = f(y1, x2)+f(y2, x1)-f(x1,
   y1)-f(x2, y2)
20 rho["S1", "S3"] = f(y1, x3)+f(y3, x1)-f(x1,
   y1)-f(x3, y3)
21 rho["S2", "S3"] = f(y2, x3)+f(y3, x2)-f(x2,
   y2)-f(x3, y3)
22 rho["S12", "S3"] = min(f(y1, x3)+f(y3, x2)-
   f(y1, x2)-f(x3, y3), f(y2, x3)+f(y3, x1)
   -f(y2, x1)-f(x3, y3))
23 #Check to see if the conditions for the
   Splice algorithm are met
24 if M == [{"x1", "y1"}, {"x2", "y2"}, {"x3", "
   y3"}] and max(rho["S1", "S2"], rho["S1
   ", "S3"]) <= rho["S2", "S3"]:
25     if rho["S1", "S3"] != rho["S12", "S3"]
       or min(rho.values()) < 0:
26         print("S1=(,x1, ...,y1, "); S2=(,
           x2, ...,y2, "); S3=(,x3, ...,
           y3, ").")
27         print("rho=", rho)
28         sys.exit("Conditions not satisfied
           by function f at trial " + str(
           cnt))
29     else:
30         cnt += 1
31 print("Run completed successfully.", cnt, "
   trials passed.")

```

Data availability

All source code and data are available online at <https://doi.org/10.5281/zenodo.11657149>.

References

- Becker, K. (2010). *Twin-constrained Hamiltonian paths on threshold graphs—an approach to the minimum score separation problem* (Ph.D. thesis), London School of Economics.
- Borges, Y., Schouerya, R., & Miyazawa, F. (2024). Mathematical models and exact algorithms for the colored bin packing problem. *Computers & Operations Research*, 164, Article 106527.
- Burkard, E., Deineko, V., van Dal, R., van der Veen, J., & Woeginger, G. (1998). Well-solvable special cases of the traveling salesman problem: A survey. *SIAM Review*, 40(3), 496–546.
- Coffman, E., Csirik, J., Galambos, G., Martello, S., & Vigo, D. (2013). *Bin packing approximation algorithms: survey and classification* (pp. 455–531). New York, NY: Springer, ISBN: 978-1-4419-7997-1, http://dx.doi.org/10.1007/978-1-4419-7997-1_35.
- Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). The MIT Press, ISBN 0262033844 9780262033848.
- Duan, R., & Pettie, S. (2014). Linear-time approximation for maximum weight matching. *Journal of the ACM*, 6(1), 1–23.
- Falkenauer, E. (1998). *Genetic algorithms and grouping problems*. John Wiley and Sons, ISBN: 9780471971504.
- Garey, M., & Johnson, D. (1979). *Computers and intractability—a guide to NP-completeness*. San Francisco: W. H. Freeman and Co., ISBN: 978-0716710455.
- Garey, M., & Johnson, D. (1981). *Approximation algorithms for bin packing problems: a survey* (pp. 147–172). Vienna: Springer Vienna, ISBN: 978-3-7091-2748-3, http://dx.doi.org/10.1007/978-3-7091-2748-3_8.
- Garraffa, M., Vancroonenburg, W., Salassa, F., Vanden Berghe, G., & Wauters, T. (2016). The one-dimensional cutting stock problem with sequence dependent cut losses. *International Transactions in Operational Research*, 23(1), 5–24.
- Gilmore, P., & Gomory, R. (1964). Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5), 655–679.
- Goulimis, C. (2004). Minimum score separation—an open combinatorial problem associated with the cutting stock problem. *Journal of the Operational Research Society*, 55, 1367–1368.
- Hawa, A., Lewis, R., & Thompson, J. (2022). Exact and approximate methods for the score-constrained packing problem. *European Journal of Operational Research*, 302(3), 847–859.
- Jansen, C. (1999). An approximation scheme for bin packing with conflicts. *Journal of Combinatorial Optimization*, 3(4), 363–377.
- Khemani, D. (2008). *A first course in artificial intelligence*. McGraw Hill India, ISBN: 978-1259029981.
- Kolmogorov, V. (2009). Blossom V: A new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1), 43–67.
- Lewis, R. (2021). *Guide to graph colouring: algorithms and applications*. Springer Cham, ISBN: 978-3-030-81056-6.
- Lewis, R. (2024). Code and data. URL.
- Lewis, R., & Holborn, P. (2017). How to pack trapezoids: Exact and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 21, 463–476.
- Lewis, R., Song, X., Dowsland, K., & Thompson, J. (2011). An investigation into two bin packing problems with ordering and orientation implications. *European Journal of Operational Research*, 213, 52–65.
- Shachnai, H., & Tamir, T. (2001). Polynomial time approximation schemes for class-constrained packing problems. *Journal of Scheduling*, 4(6), 313–338.
- Wäscher, G., Haußner, H., & Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183, 1109–1130.