Operations Research Perspectives 9 (2022) 100211

Contents lists available at ScienceDirect

Operations Research Perspectives

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

Which rectangle sets have perfect packings?

Florian Braam [a], Daan van den Berg [b,*]

[a] Energy Global Advies B.V., Rotterdam, The Netherlands
[b] Yamasan Science&Education, Amsterdam, The Netherlands

ARTICLE INFO

Keywords:
Problem structuring
Perfect rectangle packing
Solvability
Phase transition
NP-complete

ABSTRACT

In the perfect rectangle packing problem, a set of rectangular items have to be placed inside a rectangular container without overlap or empty space. In this paper, we generate a large number of random instances and decide them all with an exact solving algorithm. Both an instance's solution probability and its hardness measured in recursions or system time, seems to critically depend on t_{max}, a parameter in the generation procedure that assigns the maximally choosable random side lengths of items in the instance. We numerically characterize the solvability across instance sizes, and derive a rule for generating (un)solvable problem instances of arbitrary size.

1. Introduction

Packing problems exist in many shapes and forms, constituting a large field of research with a rich history both in academia and industry. Usually, the goal is either to optimally fill up one or more containers with objects, or to minimize the number of containers to be used. Problem parameters can vary, among which the dimensionality, the shapes of containers and objects, placement rules and temporal or economic restrictions (e.g. [1,2]).

Typically, these problems can be grouped in 'typologies', which can be considered as somewhat unstrict classification systems. Wäscher et al. building upon earlier work by Dyckhoff, present cutting problems and packing problems as two sides of the same coin: a task of arranging one or more sets of smaller items inside one or more larger objects [3,4]. Both perspectives come with their own objective measure, which therefrom go as 'input minimization' (e.g. finding the minimal number of shipping containers needed to transport a load of parcels) or 'output maximization' (e.g. getting as many parcels from the load into one or more shipping containers). Wäscher's typology also comes with the *geometric condition*, meaning that smaller items should lie entirely within the larger object after placement, and that the items cannot overlap. This typology and its geometric condition is naturally adopted by many authors in the field; it might be seen as an abstract template for many concrete problems in (industrial) operations research.

Possibly fueled by the surge in online shopping and subsequent parcel dispatchment, recent packing studies often involve 3D containers to be packed with 3D cuboid box objects, either with or without robotic assistance [5,6]. Because of the problem's high combinatorial state space, heuristic methods are often deployed [7,8], but exact approaches exist too. As for the latter, a large comparative study was conducted by Tony Wauter's team on exact output maximization in 3D container packing [9]. A particularly interesting aspect of their results, and relevant to our work, is their investigation of *scaling behavior*. As one example, they generate sets of boxes with a total volume of 70% of the container. When the number of boxes increases (an thereby the container size, to maintain the 70% volume ratio), the computational cost of the exact algorithms initially increases dramatically, but suddenly drops five orders of magnitude — at exactly 12 boxes. The authors suggest that upward from this number, instances are suddenly *solvable*, that is, regardless of their optimality, all boxes can be packed into the container under the geometric condition, whereas for instances with fewer than 12 boxes this is (almost) never the case. In this paper, we will present strong evidence that for the 2D decision problem, a similar solvability threshold exists. But in our case, it is functionally quantifiable, and indeed (also) corresponds with a peak in computational cost for an exact algorithm.

In a more general formulation, a two-dimensional instance of the rectangle packing problem consists of a set of rectangular items that should be placed in closed rectangular containers under the geometric condition. Many industrial applications exist such as pallet loading, technical design of computer chips and, with some abstraction, large scale scheduling problems as well [10–14]. Again, the two perspectives are input minimization and output maximization, and problems can be subjected to constraints of time, costs, or logistics. In some definitions of the problem, one is allowed to work on multiple containers at the same time, while in other cases one has to finish one bin before

* Corresponding author.
E-mail addresses: Florianbraam@gmail.com (F. Braam), Daan@Yamasan.nl (D. van den Berg).

https://doi.org/10.1016/j.orp.2021.100211
Received 20 October 2021; Received in revised form 13 November 2021; Accepted 28 November 2021
Available online 3 January 2022

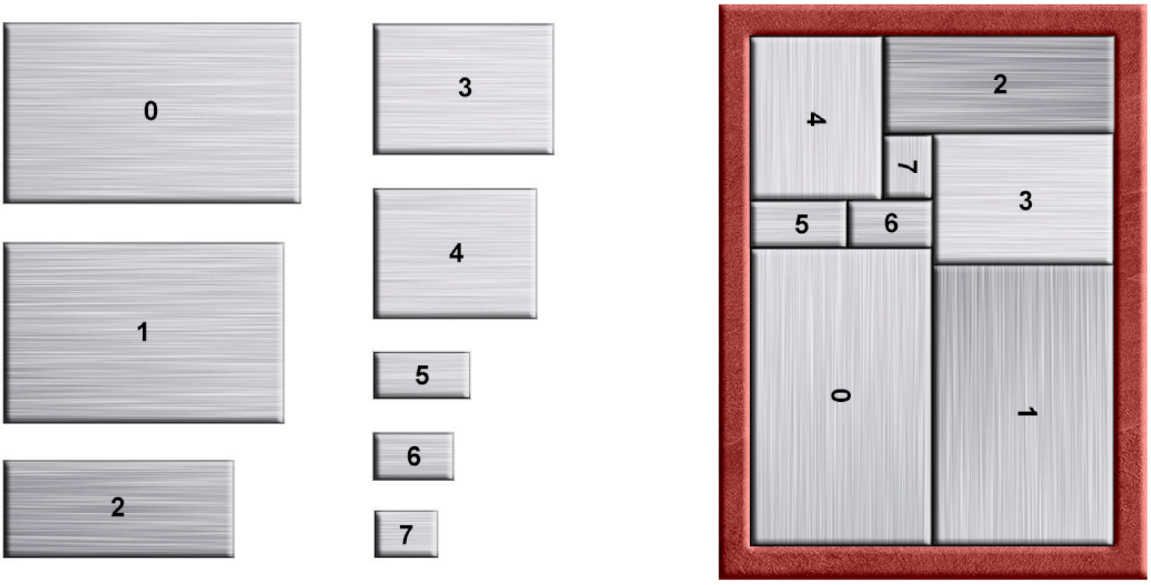

Fig. 1. Puzzle#4223 from our randomly generated dataset, having $n = 8$ tiles, labeled by their index. In this solution, tiles 1 and 3 *rectangulate* to a larger tile (right side, lower two tiles). Tiles (5 and 6) and 0 *recursively rectangulate*, tile 0 being the root (left side, lower three tiles). Exact dimensions of this puzzle can be found in the Appendix.

placing objects in a new one [15,16], especially when 'cutting stock' in industrial applications [17].

A typical instance of the latter is the ROADEF/EURO 2018 challenge posed by French glass manufacturer Saint-Gobain, in which rectangular objects need to be cut from large glass stock sheets ("jumbos") with minimal waste and additional requirements, such as guillotine cutting, order sequencing and defect avoidance. Two noteworthy applications from the same algorithmic family are an implementation of beam search and IMBA*, related to A* [18,19]. The latter of these two is particularly interesting for the optimization problem. Though it is an anytime algorithm, meaning it continually improves while it is running, it also guarantees an optimal solution in (possibly very long but) *finite time*. So despite its 'anytimeness', it somewhat leans towards a polynomial time approximation scheme (PTAS), or even a traditional exact algorithm for the problem.

For the closely related strip packing problem, in which the container is of infinite length in one dimension, approaches based on the widely used branch-and-bound paradigm can give exact solutions (See e.g. [20,21]). An interesting recent development from strip packing is the implementation of *Benders' decomposition* [22]. At its core, the technique entails slicing the items to be packed in the strip and contiguously rearranging them (see e.g. Fig. 1 from [23]).

In this paper, we will further tighten our focus, addressing the *perfect* rectangle packing problem (PRPP) with rotation, allowing rectangles to be placed either horizontally or vertically. In PRPP, the combined surface area of the rectangular objects ('tiles') equals the surface area of the container ('frame'), meaning that any solution, if existent, is a perfect fit without leftover space or overlapping tiles (Fig. 1) [24–27]. Thereby, the geometric condition from Wäscher's typology naturally holds for the PRPP, but each instance itself qualifies 'only' as a placement (or layout) problem, simply because there is just one and unselectable large object (the frame), one unselectable and ungroupable set of given small items (the tiles), and thereby also has no allocation variable. What thus remains is finding a fitting (and thereby 'optimal') arrangement only — or proving none exists.

In this sense, PRPP intuitively abrases a bit with the typology's implicit idea of 'cutting and packing', where a problem (instance) often if not always has multiple solutions, the one being better than the other. But from a computer science perspective, PRPP's characteristics firmly relegate it to a very different problem category. Since any solution to a puzzle is a tight fit, its validity can be verified by a polynomial time algorithm, even though finding one takes superpolynomial time. Furthermore, there is no difference in solution quality: any fit is an optimal fit, and the problem itself is an NP-complete decision problem [28,29]. Although the principal relation between an NP-hard optimization problem and a corresponding NP-complete decision problem is subject to a wide array of discutatory peregrinations (see for instance [30] and [31] for an elaboration on TSP)), we will not address the arguments in this paper. Instead, we will just focus on the NP-complete decision problem, the solvability of its instances, and the characteristics thereof.

In some sense, the decision problem ("can we cut these glass panes from one jumbo of identical area?") is actually a little bit easier than its cousin in optimization ("what is the lowest waste when cutting these glass panes from one or more jumbos?"). An optimization problem, in its most general shape, lacks a quick verification procedure, and is therefore NP-hard [25,32]. Second, the NP-complete class is a rather large collection of problems which are polynomially interreducible, meaning one problem can be transformed into another [33]. Centrally located in the class is the satisfiability problem, usually presented in its CNF3SAT-form, but also contains problems such as Hamiltonian cycle and k-colorability. These transformation are often nontrivial, but many of these problems share characteristics such as the complexity of the best known algorithm, or a sudden jump in solvability,[1] a *phase transition*, along some (order) parameter which usually has a constraining character. We will demonstrate these characteristics for PRPP as well, having slightly different characteristics from other NP-complete problems, raising some questions about the structure of the class and the mutuality of its members.

An important aspect of an NP-complete problem is that despite its hardness, a single instance need not be practically undecidable per se. Surely, the computation time to reach a decision increases superpolynomially in the size of the problem n, but this is only for the *best* known algorithm in a *worst case instance* ranking. For example, in the NP-complete Hamiltonian cycle problem, the difference in computation time required for different instances of $n = 32$ varies enormously. In fact, most of its graphs are extremely easily decided, but a few really

[1] Whenever we write 'solvability', we mean 'an instance('s probability of) having a solution'. The question of whether finding an instance's solution, or making sure none exists, can be done in practically feasible time, will be addressed as 'decidability'. Furthermore, we assume $P \neq NP$.

hard ones keep the problem as a whole in NP [34–36]. This variability appears to uphold for many if not most NP-complete problems, but mapping the difficulty of entire problem spaces is an open issue for different NP-complete problems [37–39]. A recurring theme in these studies is the existence of a sudden jump, a phase transition, between solvable and unsolvable instances along the increase of some parameter, such as edge degree for the Hamiltonian cycle problem, or number of clauses for CNF3SAT. But besides being a 'solvability indicator', these parameters also hold predictive properties for the computation time of corresponding instances of the problem. The easiest instances are found at both extremes of the parameter, while harder instances are found close to the solvability phase transition, even though different solvers may account for slightly different results, which might be taken as an indicator that pruning and preprocessing have, to some degree, fundamental impact on a problem's hardness [40].

In a similar but not identical way, we found a solvability phase transition in PRPP, and computational profiles to match. These conclusions come from a rather simple and straightforward experimental approach: we made ensembles of unbiased randomly generated PRPP-instances abiding by preset tile dimension restrictions (Section 2). Then, we solved them with a sophisticated exact algorithm, the details of which can be found in Section 3. After that, results will be presented in Section 4, and we will open discussions in Section 5, where some of the rather remarkable differences between our findings and other NP-complete problems will be addressed.

2. Creating puzzles

A PRPP-instance ('puzzle') consists of a rectangular frame and a tileset of n rectangular tiles, both integer-valued. Complexitywise, n is the parameter setting its upper computational bound, as well as the problem's NP-completeness. On the search space, the upper bound for instances of n tiles is $n! \cdot 2^n$, in which $n!$ expresses every possible permutation of tiles, and 2^n represents the choice in the tiles' orientations ('standing up' or 'lying down'). For this study, it is also the upper computational bound because our exact solver algorithm is of the exhaustive type — in the worst case, it has to try every combination of permutation and orientations to find a solution. Importantly though, this principle does not necessarily apply to all NP-complete problems. In the Hamiltonian cycle problem, an efficient exhaustive algorithm runs in $O(n!)$ while the problem's complexity is $O(n^2 \cdot 2^n)$. The nuance is found in the fact that the complexity portrays relatively little information about the problem as a whole, apart from being the best known upper computational bound for an exact algorithm (being the Held–Karp–Bellman for the Hamiltonian Cycle problem [41,42]).

We made 36,600 puzzles in 15 ensembles of $n = 6, 7..., 19, 20$ tiles respectively. To conserve mutual comparability for puzzles within an ensemble, we need to make sure that by design, the search space of each puzzle is no smaller than $n! \cdot 2^n$; it only makes sense to compare the number of recursions of two 17-tile puzzles, if *both* of them have an uninformed search space size of $17! \cdot 2^{17}$. There are a few technicalities to consider in order to not accidentally reduce the search space, or guarantee a priori (un)solvability. It should be noted though, that even with the restrictions below, it is possible that further properties exist which make instances easier, harder, or even (un)solvable, and uncovering these is a long-term mission of the community. For our puzzles, the following properties are guaranteed:

1. The total area of a frame is equal to the combined area of tiles in the tileset. If smaller, it is a priori unsolvable. If greater, it is not a PPRP-instance (whether solvable or not).
2. Each tile must fit inside the frame both horizontally and vertically. If it does not fit in one direction, it reduces the puzzle's search space size by a factor two. If it does not fit in either direction, the puzzle would be a priori unsolvable.
3. There are no duplicate tiles in the tileset. Having duplicate tiles reduces the factorial factor in the search space size from $n!$ to $\frac{n!}{|d_1|!|d_2|!...|d_m|!}$ for every (multi)set d_i of identical tiles. Therefore, every tile is unique for the puzzle that contains it.
4. There are no square tiles in the tileset. Square tiles can be left unrotated, and thereby the existence of square tiles reduces the factor 2^n in the search space size to 2^{n-sq} in which sq is the number of square tiles. For this reason, there are no square tiles anywhere in this study.

Table 1
Due to combinatorial reasons, the smallest possible t_{max} is slightly higher for larger n, leading to slightly smaller ensembles.

Tiles (n)	Max tile length (t_{max})	Puzzles per ensemble
6–12	6–30	2500
13–19	7–30	2400
20	8–30	2300

Because our tile generation procedure is not based on guillotining, it is far easier to first generate the tiles and then randomly assign it an eligible frame, than the other way around. The algorithm that generates the tileset has two parameters: the number of tiles in a puzzle n and the maximum side length of a tile t_{max}. The minimum side length of a tile is always 1. Puzzles are generated as follows:

1. A set of all existable tiles within the parameter range is generated. From $(1,2),(1,3)$ ..$(1,t_{max}),(2,3),(2,4)$.. up to $(t_{max}-1,t_{max})$, there are $\frac{1}{2} \cdot t_{max} \cdot (t_{max}-1)$ such tiles and by this procedure, none of them are square.
2. A lookup table for all frame dimensions from $(1,1),(2,1),(2,2),(3,1)$... up to $(n \cdot t_{max}, n \cdot t_{max})$ is created. By these dimensions, the table thereby holds all frames that could possibly be used for any puzzle of n by t_{max}.
3. From step 1, a random subset of n different tiles is selected, and its combined area is calculated.
4. The lookup table from step 2 is used to list all frames with the same area as the combined tileset. Frames that cannot accommodate all tiles from the tileset both horizontally and vertically are rejected. If there are more than 0 frames left, one is chosen at random to accompany the tileset and constitute a puzzle (otherwise, the algorithm goes back to step 3 to select n tiles anew).
5. The tileset and the frame are listed as a puzzle. Only if more puzzles are desired for the subensemble, the algorithm returns to step 3.

In this way, a subensemble of 100 unique puzzles was generated for every pair of n and t_{max}. The subensembles of all t_{max} for a single n are combined into an ensemble of puzzles with n tiles. Although it is desirable to have t_{max} cover the exact same range for all puzzle sizes n, this turned out to be impossible. The ensemble of $n = 13$ is the first of these cases; for $t_{max} = 6$ there are exactly 15 different tiles, and although $\binom{15}{13}$ gives rise to 105 tilesets, *only 49 of these have eligible frames*. For this reason, the lowest possible value of t_{max} is 7 for ensembles with $13 \leq n \leq 19$, and 8 for $n = 20$. For the sake of data homogeneity, we could have opted for all ensembles to have $8 \leq t_{max} \leq 30$, but for smaller values of n this initial part still provides some valuable information, especially when it comes to sigmoidal characterization of solvability (Section 4) so for this reason, the minimal value of t_{max} increases slightly with n (see Table 1). As a last note, we retrospectively verified the uniqueness of puzzles, and although all 36,600 puzzles are indeed unique, there are two puzzles with an identical tileset in the subensemble with $n = 18$ and $t_{max} = 7$, but these two differ by their frame dimensions.

Several rectangle packing benchmark sets have been proposed in literature (e.g. [43], and see online [44]; also see [21]), but most are

either non-decision, have small numbers of puzzles, have far too many tiles for exhaustive search, or are for other reasons inadequate for our specific needs. Furthermore, making these puzzles is a nontrivial task if one wants to avoid 'guillotining': starting with one whole tile and recursively cutting it into smaller tiles [45,46]. There are good reasons for avoiding such procedures: guillotined sets are guaranteed to have far more than zero solutions, and are thereby (much) easier for exact solvers, as a solution is easily found. Evidence from a recent study solving both types of PRPP-instances solidly demonstrates this difference [47]. Besides, in many NP-complete problems, even instances with just one (a priori guaranteed) solution are significantly easier than unsolvable instances. We hope to forward the community, and make our 36,600 instance benchmark set, and the (un)solvability of the instances, publicly available as a new benchmark set for PRPP [48].

3. Solving puzzles

Examining the solvability of PRPP-instances requires an exact (or; 'complete') search algorithm, to ensure that for each puzzle a solution will be found, should it exist. Our solver incorporates several search pruning techniques and heuristics, but it is essential to note that these are merely used for speedup, and do not compromise its exactness. So although the algorithm contains heuristics, it is not a *heuristic algorithm* as such, and is guaranteed to find a solution to a puzzle if it exists, or return "no solution" otherwise. Its traversal is based on depth-first search, an exhaustive exact search algorithm commonly used as a basis for 2D packing problems, both optimization and decision. It combines several techniques known from literature, which are referenced throughout the section, but also introduces a few new ones. It should be noted that there is no claim of algorithmic superiority here. Contrarily, we publicly supply the source code to the community for comparison, verification, and further development [49].

3.1. Placement heuristic: fail-first

The solver starts with an empty frame, placed upright and with all tiles in the tileset marked as 'available'. At each recursive step, it starts by finding the insertion point, which is the bottom-left corner of the narrowest 'valley': an empty rectangular space with tiles or frame segments on its left, right and bottom side. Note that every configuration has at least one valley, even the empty frame itself. This insertion point prioritization strategy is reminiscent, but slightly different from the often used 'bottom-left heuristic', which first finds the lowest empty space, and then picks the leftmost point therein [47,50]. Other useful heuristics for prioritizing insertion points exist, such as the newly minted 'border-first heuristic' [24].

When the insertion point is identified, available tiles are inserted following the order of the data structure, which stores the tiles horizontally from wide to narrow, breaking ties by height (see indices in Figs. 1 and 4). As tiles are stored horizontally, the first (and widest) available tile is first tried horizontally, then vertically, before progressing to the next tile. Together, prioritizing the widest tile and the narrowest valley can be seen as an instantiation of the *fail-first principle*: choosing to branch on the tightest constraints forces the solver to exclude more possibilities earlier, thereby effectively pruning more extensively higher up the search tree [51,52]. Even placing the frame upright can be thought of as implementing the fail-first principle, as unfitting partial placements along the shorter side of the frame are more readily dismissed.

When no more tiles can be placed, either the puzzle is solved or the algorithm needs to backtrack. As it tries every possible arrangement, which is bound by an ominous $n! \cdot 2^n$, being exact spells dire prospects for runtimes on a large dataset like ours, but luckily it is seldomly that bad. This is partly due to the fail-first principle, but there are also some checks that facilitate further search pruning, which we will discuss next. Steven Skiena once wrote: "Clever pruning can make short work of surprisingly hard combinatorial search problems" [53]. For this problem, he is about to be right.

3.2. Search pruning: Unfillable gap detection

Other than placement heuristics, literature provides us with a variety of checks on whether a partial solution is still completable or should be cut off and backtracked upon. An oftenly recurring principle in PRPP could be described as 'unfillable gap detection' [26,27,54]. A gap is a rectangular area of free space surrounded on three sides by tiles or frame. By our placement heuristic, a gap's open side is always either on top (a 'valley') or to the right (a 'cave'). The smaller the gap, the tighter its constraints, because fewer tiles fit, making it worthwhile to check whether a gap is potentially (un)fillable with the remaining available tiles before further recursing.

Our implementation works by creating a lookup table with an entry for every possible gap dimension of width w and height h. Behind every (w, h) - entry in the lookup table, a list of all n entries holds the potential fillup contribution for every tile to that specific valley. For example, for the lookup entry of $w = 5$ and $h = 8$, corresponding to a 40 unit valley, an available 12×9 tile's contribution is 0, because it does not fit in the valley. But an available 10×2 tile does fit in the valley when rotated upright, sticking out a bit, but still filling up 16 units of the valley, so its potential fillup contribution is 16. An available 3×2 tile fits entirely, so potentially contributes 6 to the valley's fillup. In this way, the routine adds up the potential fillup contribution for all available tiles to the (w, h) valley, and backtracks as soon as it is found to be unfillable. After checking the narrowest valley, it checks all other valleys, then checks all caves, and if none of these gaps are unfillable, recursion resumes in the bottom-left corner of the narrowest valley.

Note that this is a *negative* check: a shortage of potential fillup guarantees unsolvability of the partial configuration, but enough potential fillup *does not* guarantee it actually fits.[2] The check process runs in $O(n)$ time though, and although the memory requirements for the lookup table are relatively sizeable, the largest lookup table in this study has 136,600 entries, and is still smaller than 2 MB. Its construction happens only once per puzzle, and is done in low-polynomial time.

3.3. Full symmetry breaking: fundamental solutions

As frames are rectangular, each solution to a puzzle has exactly three full-symmetry isomorphic alternatives. By implicitly designating one of these as 'fundamental' and avoiding symmetrical branches of the search space, one can reduce computation requirements by 75%. Though saving 75% on resources is enormous in almost every real-world context, the reader should be aware that these numbers completely dwindle in the problem's growth of $O(n! \cdot 2^n)$. Still, for instances of limited sizes such as in this study, the effects just might be significant. Discerning fundamental configurations from full symmetry isomorphs is done by comparing corner tiles: as all tiles are indexed by descending size before recursing (as seen in Figs. 1 and 4), and there are at most four corner tiles, we denote a configuration as fundamental iff the corner tile with the lowest index is in the bottom-left corner of the frame. The solver verifies whenever a second, third or fourth corner tile is placed whether its index is higher than the bottom-left corner tile and backtracks it the check fails.

There is a possibility that one tile in the puzzle is wide enough to cover *both* bottom corners of the frame and in retrospect, no less than 3782 valid puzzles ($\approx 10.4\%$) have such a tile. If it is placed at the bottom two corners, the solver marks every solution as fundamental, and breaks no full symmetry. Considering this surprisingly high frequency, there might be a possibility for some improvement here, even though this phenomenon might be typical for smaller puzzles only.

[2] The acute reader may notice that this principle is similar to the requirements in the puzzle generation procedure: while the area of the tileset and the frame are equal, this does not guarantee an instance is thereby solvable — just that areawise, it is *not unsolvable*.

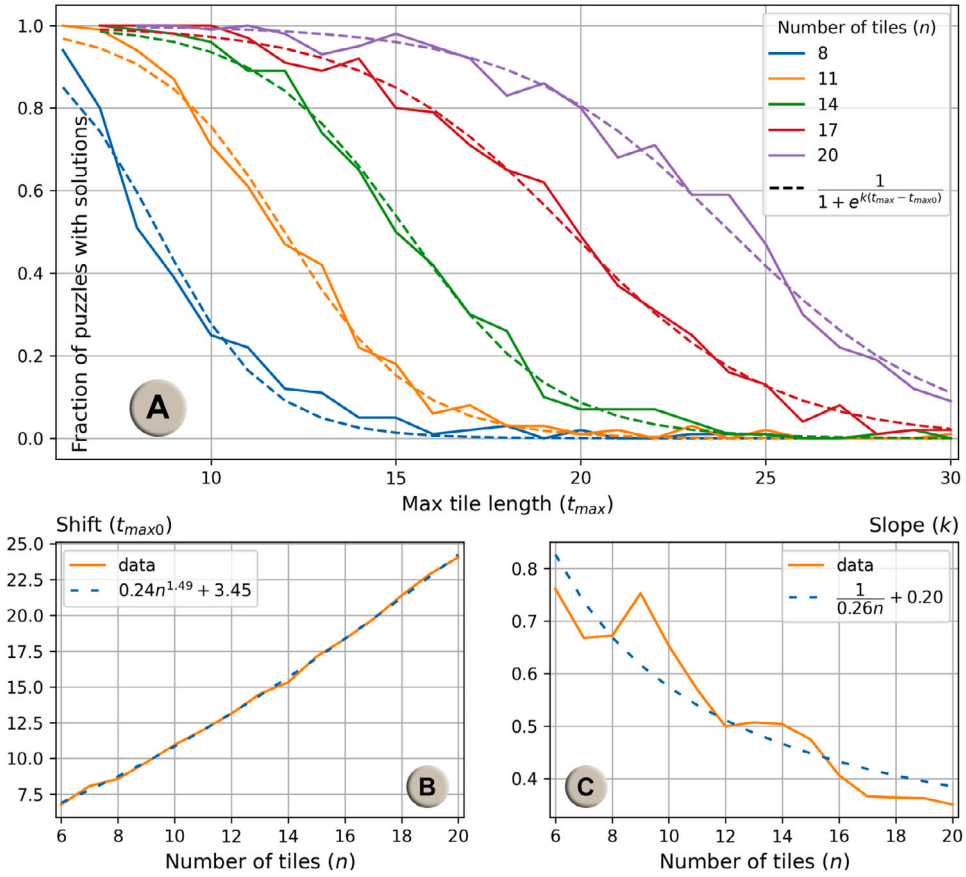

Fig. 2. For a perfect rectangle problem instance ("puzzle"), the solvability critically depends on the maximum tile side length (t_{max}) in the randomized generation procedure. The value of t_{max} for which the chance of generating a puzzle with solutions is 50% scales approximately as $O(n\sqrt{n})$ (Subfigure B).

3.4. Partial symmetry breaking: (recursively) rectangulating tiles

There is also a possibility to improve on partial symmetry in PRPP-instances. If the newly placed tile and an adjacent tile *rectangulate* (form a larger rectangle from sharing an identical side), multiple equivalent frame configurations must exist, as the rectangulating tiles could be inverted without changing the rest of the configuration. Since our placement procedure attempts lower indexed tiles before higher indexed tiles, a newly placed tile that rectangulates with an earlier placed tile should have a higher index. If the newly placed tile has a lower index than a rectangulative neighbor, the solution is clearly non-fundamental, the subtree needs to be pruned, and the solver immediately backtracks.

This procedure works well for single tile pairs, but it is also possible to apply the technique to compound rectangulations (see an example in Fig. 1), with some small adaptations to avoid a tile becoming part of multiple pairs. Rectangulations are assigned a root index equal to the maximum index of its parts. Recursively rectangulating tile pairs, when occurring, should have been placed lowest index first in any fundamental solution. Therefore, the solver backtracks if a newly placed tile rectangulates with a previously placed rectangulation whose root index is higher. Rectangulations are tracked and updated in separate subdatastructures in the puzzle's main data structure. The source code of this solver is subject to public availability on GitHub [49].

4. Results

After solving all 36,600 puzzles, there appears to be a closely sigmoidal relationship from n and t_{max} to the probability that a puzzle has a solution. For fixed n, the fraction of solvable puzzles in every 100-piece subensemble can easily be characterized with the sigmoidal function $\frac{1}{1+e^{k(t_{max}-t_{max0})}}$ through fitted values of t_{max0} and k (Fig. 2a). In this sigmoid, the parameter k is the steepness (or 'suddenness') of the slope, whereas t_{max0} is the horizontal shift. By the functional definition therefore, t_{max0} designates the value of t_{max} at which the puzzle generator should be parameterized to generate 'unguessable' puzzles, that have a 50% chance[3] of having one or more solutions.

We separately fitted the sigmoid for all ensembles $6 \leq n \leq 20$, with fit-errors $0.001 \leq MSE \leq 0.004$ across the entirety, but for the sake of clarity we only show $n \in \{8, 11, 14, 17, 20\}$ in Fig. 2a. It should (again) be noted that for larger n, characterizations start slightly more to the right, having a slightly shorter domain, stemming from combinatorial reasons given in Section 2.

Parameter k, as found in the sigmoidal fits, can be characterized itself as $k = \frac{1}{0.26n} + 0.20$, therewith expressing the steepness of the curve through n (Fig. 2c). As with most sigmoidal phenomena, it unsteepens through n ($MSE < 0.003$). Possibly the most important result of this study however, is the shift of the sigmoidal inflection point t_{max0} through instance size n (Fig. 2b). It gives the value of t_{max} for which a puzzle has exactly 50% chance of being solvable. Its value scales polynomially in n as $0.24 \cdot t_{max0} = n^{1.49} + 3.45$, thereby being very close to $t_{max0} = O(n\sqrt{n})$ (with $MSE < 0.03$).

In terms of making puzzles, these values have very direct consequences. It means that to create an unguessable 8-tile puzzle, t_{max} should be 9. For 14 tiles, the value of t_{max} needs to be 16 and for 20 tiles, set $t_{max} = 24$ to get unguessable puzzles. And the functional characterization also allows us some hypothetical forward projection

[3] These values are rounded to integers for implementation measures, so in practice the solution probability should be read as *very near* 50%.

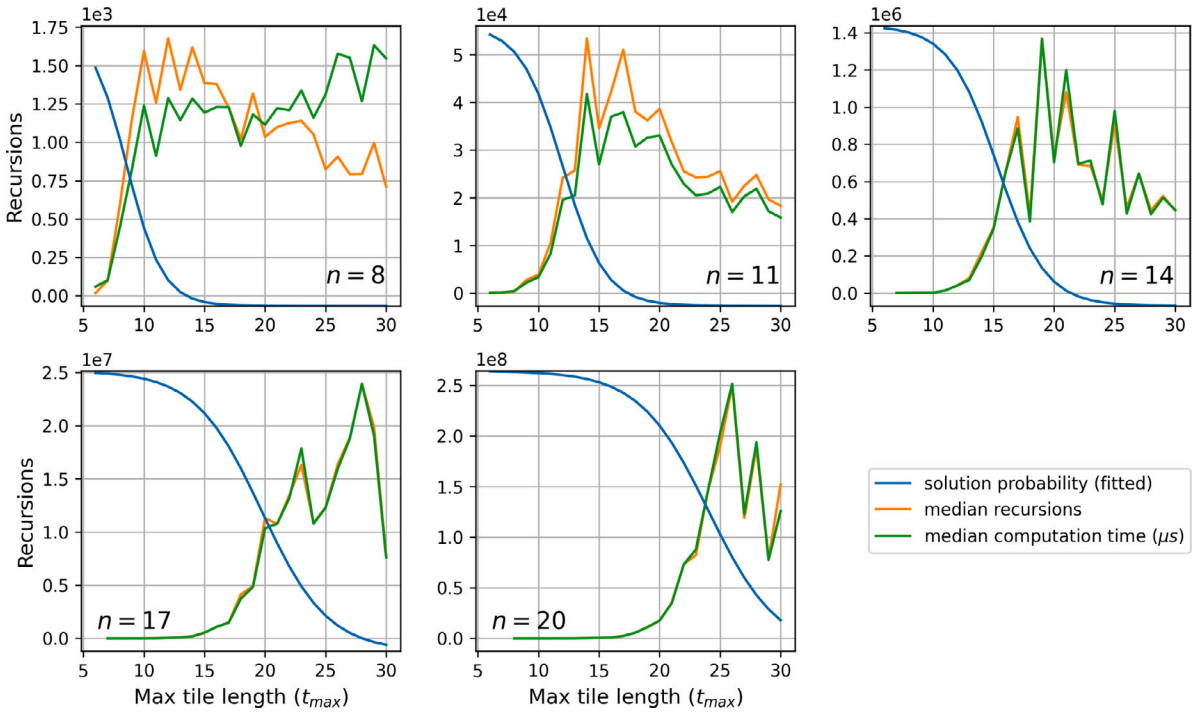

Fig. 3. Typical for NP-complete problems, the hardest puzzles are found just after the bend in the solvability phase transition. For smaller puzzles, recursion time and real time show inverted trends, but measurements become synchronized as puzzle size increases.

Table 2
The hardest subensembles for puzzles with n tiles measured in either median or total number of recursions, are found exclusively above the solvability point. The single hardest puzzle however can be found virtually anywhere across the t_{max} value.

Tiles (n)	50% point	Total	Median	Single puzzle
8	8.58	10 (+17%)	12 (+40%)	10 (+17%)
11	11.99	14 (+17%)	14 (+17%)	11 (−8%)
14	15.31	17 (+11%)	19 (+24%)	11 (−22%)
17	19.72	25 (+27%)	28 (+42%)	27 (+37%)
20	24.05	26 (+8%)	26 (+8%)	26 (+8%)

too. For a puzzle with 50 tiles, set t_{max} to 85. For 100 tiles, take t_{max} = 233, for 500 tiles, choose t_{max} = 2538 and for a puzzle of 1000 tiles, t_{max} needs to be 7132 to be unguessable. These numbers assume that valid frames exist for all tilesets generated with these values but also holds a vexing paradox: even if these projections are accurate, there is not an algorithm in the world that even comes close to solving puzzles of these sizes, so results could never realistically be verified. Still, it is a first serious estimate of how to make typically (un)guessable puzzles of those magnitudes. When combined with the characterization of k, puzzles of any size with any solvability probability can be made. Again, these are principally unverifiable, except maybe for very high and very low-probability cases, as other NP-complete problems show that instances far away from the solvability phase transition's inflection point are, on average, easier to solve.

Instance hardness is typically expressed in recursions and gives a largely system-independent measure that will 'only' scale up or down a constant factor when a different platform, different machine, different compiler or a different programming language is used. Though rigorous, it tells us very little about the practical time needed for solving puzzles such as ours. Measuring system time, although subject to all these inconsistencies, does provide a bit of foothold. In this study we measured both, and runs were done on a single thread of an Intel Xeon E5-2650V2 with a base clock of 2.6 GHz and a maximum turbo of 3.40 GHz.

Initially, these two measures show very different patterns, but they become ever more synchronized as puzzles get larger (Fig. 3). This is certainly not true for all algorithms on all NP-complete problems [35], and has a lot to do with the exact optimization procedures, pruning, and preprocessing, but the synchronicity allows us to stick to recursions for now. In this experiment, the hardest instances were found slightly right of the solvability phase transition (Fig. 3), although the exact locus depends somewhat on whether a subensemble's median or total number of recursions is taken as a hardness measure (Table 2). The subensemble requiring the highest median number of recursions is located at a higher or equal t_{max} value to the subensemble requiring the highest total number of recursions in 13 out of 15 ensembles. All of these values are 8% to 45% higher than the inflection point of 50% solvability. The location of the single hardest puzzle in a subensemble proved slightly more elusive, in t_{max} values ranging from −36% to 107% relative to the inflection point. Thereby, these hard single instances are usually located to the right of the phase transition, but occasionally also to the left (Table 2).

As with most NP-complete problems, the hardest puzzles *without* solutions are slightly harder than the hardest puzzles *with* solutions. These are usually the rare cases where little or no pruning can be done, and the algorithm runs down the entire state space. For cases which are unprunable but *do* have solutions, the algorithm halts as soon as it finds one, leaving the remainder of the search space untouched.

5. Conclusion & discussion

The first and most important (and unanswered) question emanating from these results is whether the inverse of our results holds: is it possible, for a given rectangle puzzles of arbitrary size, to *accurately guess its solution probability without solving it* based on its rectangles alone? We believe it is. But because of the high unlikeliness that puzzles of substantial size (e.g. such as found in industry) will be solvable any time soon, answers are more likely to come from theory of NP-completeness than from experimental solving such as done in this study. Notably, the t_{max} parameter is stochastic, and the maximum side length did not reach the t_{max} value in 8437 puzzles (23.1%). Even though the top half of these occurred in the extremer puzzles, with few tiles and high maximum side lengths ($n \approx 8.74, t_{max} \approx 24.3$), a

Fig. 4. This study's hardest puzzle from the $n = 20$ ensemble. Is it possible to predict whether it has a solution, and in how much time it can be decided, from the frame and tiles' characteristics only? Exact dimensions for tiles and frame can be found in the Appendix.

recently published replication study on an asymmetric traveling salesman problem shows that there can be a significant difference between a priori and a posteriori classification [30]. That is unlikely in our case but possibly and preferably, a future investigation into per-instance classification could provide a sharper resolution on these details.

In a broader scientific context, the sigmoidal phase transition in the solvability of PRPP-instances does not come as a complete surprise. Other NP-complete problems like k-colorability, CNF3SAT and Hamiltonian cycle all have similarly shaped phase transitions, or in Ian Gent's words:"[we have yet to find an NP-complete problem that *lacks* a phase transition]" [39]. Usually, these solvable-to-unsolvable transitions can be linked to some parameter increase.[4] For k-colorability, the probability of a solution suddenly drops with the increasing ratio of constraints-per-variable [37]. For CNF3SAT, the clauses-to-variables ratio $\alpha \approx 4.26$ predicts whether random formulae have solutions [38,55,56]. For Hamiltonian cycle, it is the vertex degree that functions as the indicator of solvability; for v vertices, graphs with edge degrees below $ln(v) + ln(ln(v))$ almost never have a Hamiltonian cycle, where graphs above it are almost surely Hamiltonian [24,57]. Even the routability of netlists in VLSI, a stacked optimization problem, very different from PRPP, behaves strictly sigmoidal in its routability, with the length of its netlist being the predictive parameter, scaling as $O(ln(M))$ in its mesh size M [58]. And results from this study suggest yet another scaling law for solvability in an NP-complete problem, being $t_{max} \approx O(n\sqrt{n})$. The point here is, that even as NP-complete problems are interreducable, and similar solvability characteristics could be expectable, the phase transitions for these problems are very different, raising questions about the nature of the reduction mappings themselves.

An important technical sidenote to be made for these results. Many sigmoidal functions can be fit to this data and all have relatively good fit, but the symmetric sigmoid function we used here simply had the lowest MSE. It is substantially different from the *analytically derived* and *asymmetric* function $e^{-e^{-2c}}$ that predicts the Hamiltonicity of a randomly generated graph [57]. In this light, it is important to mention that larger puzzles ($n = 18, 19, 20$) actually fit better in the asymmetric sigmoid. So it is possible, that PRPP-problem has the same phase transition as the Hamiltonian cycle problem, but that these effects are only noticeable on a (much) larger scale. This is as yet untestable because of the higher complexity of the PRPP-problem, but it would make sense from a theoretical point of view, which dictates that NP-complete problems are mutually reducible.

But even though these sigmoidal phase transitions are quite ubiquitous among NP-complete problems, their mutuality is far from trivial. Or (again in Ian Gent's words): "[Although any NP-complete problem can be transformed into any other NP-complete problem, this mapping does *not* map the problem space uniformly]"). One way to think of it is this: the ensemble of all possible CNF3SAT formulae with 30 variables undergoes a solvability phase transition as the number of its clauses increases. The Hamiltonian cycle problem (for say, 80 vertices) *also* undergoes a solvability phase transition, in this case through the vertex degree. There are several proofs of NP-completeness that transform any CNF3SAT-formula into a Hamiltonian cycle problem instance, so it makes sense to think that the phase transition in CNF3SAT finds a linearly projected disposition in Hamiltonian cycle [29,59,60]. But this is not trivially so. For CNF3SAT, it is the number of atomic variables that controls its complexity, and thereby its NP-completeness. For Hamiltonian cycle, it is the number of vertices that define its non-polynomial runtime complexity. But the number of vertices in the Hamiltonian cycle problem instance obtained from a transformed CNF3SAT-instance depends on the number of *clauses* instead of the number of variables. One instance of CNF3SAT with 30 variables might end up as a Hamiltonian cycle problem instance with 80 vertices, while another 30-variable CNF3SAT-instance might result in a 144-vertex problem instance. This means that any projection of a phase transition in an ensemble of CNF3SAT-instances smears out across different ensembles of Hamiltonian cycles problem instances, making transformations non-trivial to say the least. It is even possible that different transformations

[4] Sometimes called an 'order parameter'.

lead to different projections. Having four NP-complete problems with four different solvability scaling laws might say something about these projections. Is it possible to designate projections as canonical? Or even minimal?

The 'hardness peak' from Fig. 3 and Table 2 comes with its own debatable properties. It is true that the hardest PRPP-puzzles reside very near the solvability phase transition, just like in k-colorability, just like in CNF3SAT, and just like in the Hamiltonian cycle problem, for which similar results even hold for a wide variety of exact algorithms [35]. But a recent study has also shown that a targeted evolutionary algorithm for creating hard Hamiltonian cycle problem instances finds graphs that are approximately 10^{15} times harder than those found in large randomized ensembles[5] [36,61]. The point is: in at least one other NP-complete problem, instances exist that are several magnitudes harder than anything that turns up in a randomly generated ensemble. An explanation for this phenomenon stems from the fact that these instances are extremely rare, possibly because they are *structured*, meaning the have low Kolmogorov complexity. But as an inconclusive counter indication from two studies in Euclidean TSP, low Kolmogorov complexity appears to be associated with both easy and hard instances [62,63].

We do not exactly know *how* rare low Kolmogorov complexity hard and easy instances are, but if the complexity is indeed the discerning criterion, the universe might not be large enough to ever produce one in a random ensemble. Our PRPP-puzzles are NP-complete, also come in randomized ensembles, and therefore these earlier studies could suggest that, possibly very far away from the solvability phase transition, puzzles exist that are much much harder than anything we found in this study. Whatever they look like, we can only guess. A practical problem for an evolutionary approach is that for these puzzles, a mutation is not so easily made. However, the plant propagation algorithm which was used in the Hamiltonian cycle study has been used on a great variety of constrained NP-hard problems as yet [64,65] and shown quite some consistent and parameter-robust behavior [66–68]. It might, with some adaptations and transformations, also be able to find extremely hard PRPP-puzzles in extremely unlikely places. For now however, this item remains on the ever growing ToDo-list of future work.

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.

Acknowledgments

Many thanks to Joeri Sleegers, first author of several Hamiltonian cycle studies (cited in situ), for helping with the assessment in the last paragraph of the paper.

Appendix. Dimensions of puzzles from this paper

The puzzle from Fig. 1 has a frame of 22×31, and tiles of 18×11, 17×11, 14×6, 11×8, 10×8, 6×3, 5×3, 4×3. The puzzle from Figure ?? has a frame of 55×57, and tiles of 30×6, 29×18, 29×12, 27×4, 26×5, 26×3, 25×5, 23×8, 23×5, 22×1, 21×15, 19×11, 18×11, 17×15, 17×5, 14×9, 11×7, 8×1, 6×5, 5×4. Community challenge: solve this problem. Hand in your solution. The first candidate that submits a solution *at least* as good as ours (measured in number of placed tiles) gets a certificate. Students get an extra prize. Puzzle might not be completely solvable.

[5] Extrapolated data from [35,36], and they are very far away from the phase transition! In a randomized ensemble of 9920 graphs with 32 vertices (20 for every possible degree), the hardest graph for the best known exact algorithm (Vacul's algorithm) takes nearly 10^5 recursions. In a targeted search for size 8 to 14, the hardest graph requires over 3 million recursions, but the trend is quite tight, and when functionally extended to a graph of 32 vertices, will require approximately $4 \cdot 10^{20}$ recursions.

References

[1] Gendreau M, Iori M, Laporte G, Martello S. A tabu search algorithm for a routing and container loading problem. Transp Sci 2006;40(3):342–50.
[2] Castellucci PB, Toledo FM, Costa AM. Output maximization container loading problem with time availability constraints. Oper Res Perspect 2019;6:100126.
[3] Wäscher G, Haußner H, Schumann H. An improved typology of cutting and packing problems. European J Oper Res 2007;183(3):1109–30.
[4] Dyckhoff H. A typology of cutting and packing problems. European J Oper Res 1990;44(2):145–59.
[5] Li H, Wang Y, Ma D, Fang Y, Lei Z. Quasi-Monte-Carlo tree search for 3D bin packing. In: Chinese conference on pattern recognition and computer vision. Springer; 2018, p. 384–96.
[6] Edelkamp S, Gath M, Rohde M. Monte-Carlo tree search for 3D packing with object orientation. In: Joint German/Austrian conference on artificial intelligence (KÜNstliche Intelligenz). Springer; 2014, p. 285–96.
[7] Parreño F, Alvarez-Valdes R, Oliveira JF, Tamarit JM. A maximal-space algorithm for the container loading problem. INFORMS J Comput 2008;20(3):412–22.
[8] Correcher JF, Alonso MT, Parreño F, Alvarez-Valdes R. Solving a large multi-container loading problem in the car manufacturing industry. Comput Oper Res 2017;82:139–52.
[9] Silva EF, Toffolo TÂM, Wauters T. Exact methods for three-dimensional cutting and packing: A comparative study concerning single container problems. Comput Oper Res 2019;109:12–27.
[10] Dowsland KA, Dowsland WB. Packing problems. European J Oper Res 1992;56(1):2–14.
[11] Alpert CJ, Mehta DP, Sapatnekar SS. Handbook of algorithms for physical design automation. CRC Press; 2008.
[12] Leung JY. Handbook of scheduling: Algorithms, models, and performance analysis. CRC Press; 2004.
[13] Caramia M, Giordani S, Iovanella A. Grid scheduling by on-line rectangle packing. Networks Int. J. 2004;44(2):106–19.
[14] Simonis H, O'Sullivan B. Search strategies for rectangle packing. In: International conference on principles and practice of constraint programming. Springer; 2008, p. 52–66.
[15] Jylänki J. A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing. 2010, Retrived February 20, 2021 from http://Clb.Demon.Fi/Files/RectangleBinPack.Pdf, Citeseer, Possibly not peer-reviewed.
[16] Martello S, Vigo D. Exact solution of the two-dimensional finite bin packing problem. Manage Sci 1998;44(3):388–99.
[17] Hopper E, Turton BC. A review of the application of meta-heuristic algorithms to 2D strip packing problems. Artif Intell Rev 2001;16(4):257–300.
[18] Parreño F, Alonso MT, Alvarez-Valdes R. Solving a large cutting problem in the glass manufacturing industry. European J Oper Res 2020;287(1):378–88.
[19] Libralesso L, Fontan F. An anytime tree search algorithm for the 2018 ROADEF/EURO challenge glass cutting problem. European J Oper Res 2021;291(3):883–93.
[20] Arahori Y, Imamichi T, Nagamochi H. An exact strip packing algorithm based on canonical forms. Comput Oper Res 2012;39(12):2991–3011.
[21] Iori M, de Lima VL, Martello S, Miyazawa FK, Monaci M. Exact solution techniques for two-dimensional cutting and packing. European J Oper Res 2021;289(2):399–415.
[22] Delorme M, Iori M, Martello S. Logic based benders' decomposition for orthogonal stock cutting problems. Comput Oper Res 2017;78:290–8.
[23] Côté J-F, Dell'Amico M, Iori M. Combinatorial benders' cuts for the strip packing problem. Oper Res 2014;62(3):643–61.
[24] van den Berg D, Braam F, Moes M, Suilen E, Bhulai S. Almost squares in almost squares: Solving the final instance. Data Anal 2016 2016;81.
[25] Lodi A, Martello S, Monaci M. Two-dimensional packing problems: A survey. European J Oper Res 2002;141(2):241–52.
[26] Lesh N, Marks J, McMahon A, Mitzenmacher M. Exhaustive approaches to 2D rectangular perfect packings. Inform Process Lett 2004;90(1):7–14.
[27] Hougardy S. A scale invariant algorithm for packing rectangles perfectly. In: Proceedings of the fourth international workshop on bin packing and placement constraints. 2012.
[28] Korf RE. Optimal rectangle packing: Initial results. In: ICAPS. 2003. p. 287–95.
[29] Garey MR, Johnson DS. Computers and intractability, vol. 174. freeman San Francisco; 1979.
[30] Sleegers J, Olij R, van Horn G, van den Berg D. Where the really hard problems aren't. Oper Res Perspect 2020;7:100160.
[31] Applegate DL, Bixby RE, Chvátal V, Cook WJ. The traveling salesman problem. Princeton University Press; 2011.

[32] Alvarez-Valdes R, Parreño F, Tamarit JM. A branch and bound algorithm for the strip packing problem. OR Spectrum 2009;31(2):431–59.
[33] Cook SA. The complexity of theorem-proving procedures. In: Proceedings of the third annual ACM symposium on theory of computing. ACM; 1971, p. 151–8.
[34] van Horn G, Olij R, Sleegers J, van den Berg D. A predictive data analytic for the hardness of Hamiltonian cycle problem instances. Data Anal 2018 2018;101.
[35] Sleegers J, van den Berg D. Backtracking (the) algorithms on the Hamiltonian cycle problem. Int J Adv Intell Syst 2021;14. (in press; arXiv-preprint available at https://arxiv.org/abs/2107.00314).
[36] Sleegers J, van den Berg D. Looking for the hardest Hamiltonian cycle problem instances. In: IJCCI. 2020. p. 40–8.
[37] Cheeseman PC, Kanefsky B, Taylor WM. Where the really hard problems are. In: IJCAI, vol.91. 1991. p. 331–340.
[38] Hayes B. Computing science: Can't get no satisfaction. Am Sci 1997;85(2):108–12.
[39] Gent IP, Walsh T. The TSP phase transition. Artificial Intelligence 1996;88(1–2):349–58.
[40] Aguirre ASM, Vardi M. Random 3-SAT and BDDs: The plot thickens further. In: International conference on principles and practice of constraint programming. Springer; 2001, p. 121–36.
[41] Held M, Karp RM. A dynamic programming approach to sequencing problems. J Soc Indust Appl Math 1962;10(1):196–210.
[42] Bellman R. Combinatorial processes and dynamic programming. Technical repor, The RAND Corporation, 1700 Main St., Santa Monica, California; 1958.
[43] Iori M, De Lima V, Martello S, M. M. 2DPackLib: A two-dimensional cutting and packing library. Optim Lett 2021. http://dx.doi.org/10.1007/s11590-021-01808-y.
[44] 2DPackLib. 2DPackLib.
[45] Hopper E, Turton B. Problem generators for rectangular packing problems. Stud Inform Univ 2002;2(1):123–36.
[46] Wang PY, Valenzela CL. Data set generation for rectangular placement problems. European J Oper Res 2001;134(2):378–91, (The second author's name "Valenzuela", is misspelled in the paper's header)..
[47] Pejic I, van den Berg D. Monte Carlo tree search on perfect rectangle packing problem instances. In: Proceedings of the 2020 genetic and evolutionary computation conference companion. 2020. p. 1697–703.
[48] Braam F. Publicly accessible PRPP-benchmark set with 36,600 instances. 2021, GitHub Repository, GitHub, https://github.com/flobrm/perfect-rectangle-packing-solvability-data.
[49] Braam F. Publicly accessible source code. 2021, GitHub Repository, GitHub, https://github.com/flobrm/perfect-rectangle-packer.
[50] Chazelle B. Efficient implementation. IEEE Trans Comput 1983;32(8):697–707.
[51] Beck JC, Prosser P, Wallace RJ. Trying again to fail-first. In: International workshop on constraint solving and constraint logic programming. Springer; 2004, p. 41–55.
[52] Haralick RM, Elliott GL. Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence 1980;14(3):263–313.
[53] Skiena SS. The algorithm design manual. Springer Science & Business Media; 1998, p. 247.
[54] Kenmochi M, Imamichi T, Nonobe K, Yagiura M, Nagamochi H. Exact algorithms for the two-dimensional strip packing problem with and without rotations. European J Oper Res 2009;198(1):73–83.
[55] Larrabee T, Tsuji Y. Evidence for a satisfiability threshold for random 3CNF formulas. Citeseer; 1992.
[56] Selman B, Mitchell DG, Levesque HJ. Generating hard satisfiability problems. Artificial Intelligence 1996;81(1–2):17–29.
[57] Komlós J, Szemerédi E. Limit distribution for the existence of Hamiltonian cycles in a random graph. Discrete Math 1983;43(1):55–63.
[58] Jansen R, Vinkesteijn Y, van den Berg D. On the solvability of routing multiple point-to-point paths in manhattan meshes. In: Proceedings of the 2020 genetic and evolutionary computation conference companion. 2020. p. 1685–9.
[59] Mitchell J. Hamiltonian cycle is NP-complete. 2021, (possibly not peer reviewed; consulted November 11th 2021), https://bit.ly/3hLH9GU.
[60] Papadimitriou CH, Steiglitz K. Combinatorial optimization: Algorithms and complexity. Courier Corporation; 1998.
[61] Sleegers J, van den Berg D. Plant propagation & hard Hamiltonian graphs. In: Evo* late breaking abstracts. 2020. p. 10–4.
[62] Hougardy S, Zhong X. Hard to solve instances of the euclidean traveling salesman problem. 2018, arXiv preprint arXiv:1808.02859.
[63] Fischer T, Stützle T, Hoos H, Merz P. An analysis of the hardness of TSP instances for two high performance algorithms. In: Proceedings of the sixth metaheuristics international conference. 2005. p. 361–367.
[64] Geleijn R, van der Meer M, van der Post Q, van den Berg D. The plant propagation algorithm on timetables: First results. In: EVO* 2019. 2019. p. 2.
[65] Paauw M, Van den Berg D. Paintings, polygons and plant propagation. In: International conference on computational intelligence in music, sound, art and design (Part of EvoStar). Springer; 2019, p. 84–97.
[66] De Jonge M, van den Berg D. Plant propagation parameterization: Offspring & population size. In: Evo* 2020. 2020. p. 19.
[67] De Jonge M, van den Berg D. Parameter sensitivity patterns in the plant propagation algorithm. In: IJCCI. 2020. p. 92–99.
[68] Vrielink W, van den Berg D. Fireworks algorithm versus plant propagation algorithm. Proceedings of ECTA 2019 2019;101–12.