Theoretical Computer Science 839 (2020) 13–20

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On the exact complexity of polyomino packing

Hans L. Bodlaender [a,b,1], Tom C. van der Zanden [a,*]

[a] Department of Computer Science, Utrecht University, Utrecht, the Netherlands
[b] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, the Netherlands

ARTICLE INFO

Article history:
Received 10 October 2018
Received in revised form 18 July 2019
Accepted 15 May 2020
Available online 28 May 2020

Keywords:
Polyomino packing
Exact complexity
Exponential time hypothesis

ABSTRACT

We show that the problem of deciding whether a collection of polyominoes, each fitting in a $2 \times O(\log n)$ rectangle, can be packed into a $3 \times n$ box does not admit a $2^{o(n/\log n)}$-time algorithm, unless the Exponential Time Hypothesis fails. We also give an algorithm that attains this lower bound, solving any instance of polyomino packing with total area n in $2^{O(n/\log n)}$ time. This establishes a tight bound on the complexity of Polyomino Packing, even in a very restricted case. In contrast, for a $2 \times n$ box, we show that the problem can be solved in strongly subexponential time.

1. Introduction

The complexity of games and puzzles is a widely studied topic, and the complexity of most games and puzzles in terms of completeness for a particular complexity class (NP, PSPACE, EXPTIME, ...) is generally well-understood (see e.g. [1] for an overview). Results in this area are not only mathematically interesting and fun, but are also a great educational tool for teaching hardness reductions. However, knowing that a game or puzzle is NP-complete does not provide a very detailed picture: it only tells us that there is unlikely to be a polynomial-time algorithm, but leaves open the possibility that there might be a very fast superpolynomial but subexponential-time algorithm. This issue was precisely the motivation for introducing the Exponential Time Hypothesis [2].

The Exponential Time Hypothesis (ETH) states that there exists no algorithm solving n-variable 3-SAT in $2^{o(n)}$ time. Assuming this hypothesis, and by designing efficient reductions (that do not blow up the instance size too much), it is possible to derive conditional lower bounds on the running time of an algorithm.

In this paper, we study the Polyomino Packing problem from the viewpoint of exact complexity. We give a reduction from 3-SAT, showing that Polyomino Packing can not be solved in $2^{o(n/\log n)}$ time, even if the target shape is a $3 \times n$ rectangle and each piece fits in a $2 \times O(\log n)$ rectangle. As the reduction is self-contained, direct from 3-SAT and rather elegant, it could be an excellent example to use for teaching. We also show that this is tight: Polyomino Packing can be solved in $2^{O(n/\log n)}$ time for any set of polyominoes of total area n that have to be packed into any shape.

Polyomino Packing appears to behave similarly to Subgraph Isomorphism on planar graphs, which has exact complexity $2^{\Theta(n/\log n)}$ [3] (i.e., there exists an algorithm solving the problem in $2^{O(n/\log n)}$ time on n-vertex graphs, and unless the ETH fails there is no $2^{o(n/\log n)}$-time algorithm).

Demaine and Demaine [4] showed that packing n polyominoes of size $\Theta(\log n) \times \Theta(\log n)$ into a square box is NP-complete. This result left open a gap, namely of whether the problem remained NP-complete for polyominoes of *area*

* Corresponding author.
E-mail addresses: H.L.Bodlaender@uu.nl (H.L. Bodlaender), T.vanderZanden@maastrichtuniversity.nl (T.C. van der Zanden).

1 This work was supported by the NETWORKS project, funded by the Netherlands Organization for Scientific Research NWO [project no. 024.002.003].

https://doi.org/10.1016/j.tcs.2020.05.025
0304-3975/

$O(\log n)$. This gap was recently closed by Brand [5], who showed that Polyomino Packing is NP-complete even for polyominoes of size $3 \times O(\log n)$ that have to be packed into a square. However, Brand's construction effectively builds up larger (more-or-less square) polyominoes by forcing smaller (rectangular) polyominoes to be packed together in a particular way, by using jagged edges that correspond to binary encodings of integers to enforce that certain pieces are placed together.

Our reduction also uses binary encoding of integers to force that various pieces are placed together. However, in contrast, it gives hardness for a much more restricted case (packing polyomino pieces of size $2 \times O(\log n)$ into a rectangle of height 3) and also reduces directly from 3-SAT, avoiding the polynomial blowup incurred by Brand's reduction from 3-Partition, thus giving a tight (under the Exponential Time Hypothesis) lower bound. As 3-Partition is a frequently used tool for showing hardness of various types of packing puzzles and games, we believe that these techniques could be used to give (tight, or at least strong) lower bounds on the complexity of other games and puzzles.

This result is tight in another sense: we show that Polyomino Packing where the target shape is a $2 \times n$ rectangle admits a $2^{O(n^{3/4} \log n)}$-time algorithm, so $3 \times n$ is the smallest rectangle in which a $2^{\Omega(n/\log n)}$-time lower bound can be attained.

Note that our results are agnostic to the type (free, fixed or one-sided) of polyomino used. That is, it does not matter whether we are able to rotate (one-sided), rotate and flip (free) or not (fixed) our polyominoes. Our reduction creates instances whose solvability is preserved when changing the type of polyomino, while the algorithms can easily be adapted to work with any type of polyomino. In the following, we consider the Polyomino Packing problem, which asks whether a given set of polyominoes can be packed to fit inside a given target shape. If we include the additional restriction that the area of the target shape is equal to the total area of the pieces, we obtain the Exact Polyomino Packing problem.

2. Lower bounds

Theorem 1. *Unless the Exponential Time Hypothesis fails, there exists no $2^{o(n/\log n)}$-time algorithm for* Polyomino Packing, *even if the target shape is a $3 \times n$ box, the bounding box of each polyomino is of size $2 \times O(\log n)$ and each polyomino has area $\Theta(\log n)$.*

Proof. A weaker version of the statement follows by a simple reduction from the Orthogonal Vector Crafting problem [6]. However, because obtaining the bound on the piece size requires a deeper understanding of the proof, and to illustrate the technique, we give a self-contained proof that closely follows the presentation of [6].

We proceed by reduction from n-variable 3-SAT. The Exponential Time Hypothesis, together with the Sparsification Lemma [7], implies (conditionally) that this problem does not admit a $2^{o(n)}$-time algorithm, even if the number of clauses $m = O(n)$.

Using the following well-known construction, we can furthermore assume that each variable occurs as a literal at most 3 times: replace each variable x_i that occurs $k > 3$ times by k new variables $x_{i,1}, \ldots, x_{i,k}$ and add the clauses $(\neg x_{i,1} \vee x_{i,2}) \wedge (\neg x_{i,2} \vee x_{i,3}) \wedge \ldots \wedge (\neg x_{i,k-1} \vee x_{i,k}) \wedge (\neg x_{i,k} \vee x_{i,1})$. This only increases the total number of variables and clauses linearly (assuming we start with a linear number of clauses).

We remark that our construction works for general SAT formulas. The Sparsification Lemma is only needed to achieve the stated $2^{\Omega(n/\log n)}$ lower bound, and the bound on the number of occurrences of a variable is only needed to obtain the bound on the piece size.

Our construction will feature three types of polyomino: n *formula-encoding polyominoes*, n *variable-setting* polyominoes and m *clause-checking* polyominoes. We number the variables of the input formula $1, \ldots, n$ and the clauses $n+1, \ldots, n+m$. With every clause or variable we associate a bitstring of length $22 + 4\lceil \log(n+m) \rceil$, which is obtained by taking the binary representation of that clause/variable's number, padding it with 0's to obtain a bitstring of length $\lceil \log(n+m) \rceil$, replacing every 0 by 01 and every 1 by 10 (thus ensuring the number of 1's in the bitstring is equal to the number of 0's, and that the bitstring contains at most 2 consecutive zeroes or ones) and then appending a reversed copy of the bitstring to itself (making it palindromic). Finally, we prepend 11110001111 and append 11110001111 (note that thus the start and end of the bitstring is the only place to feature 3 or more consequitive 0's).

For any bitstring, we can create a *corresponding polyomino*: given a bitstring of length k, its corresponding polyomino fits in a $2 \times k$ rectangle, whose top row consists of k squares, and whose bottom row has a square whenever the bitstring has a 1 in that position. For each such polyomino, we can also create a *complementary polyomino* that mates with it to form a $3 \times k$ rectangle (which can also be seen as a flipped version of the polyomino corresponding to the complement of the bitstring, i.e., the bitstring with all zeroes replaced by ones and vice-versa). Fig. 1 shows several example corresponding polyominoes and their complements. Note that since the bitstrings are palindromic, the thus created polyominoes are achiral, i.e., invariant over being flipped.

We can *concatenate* two polyominoes corresponding to bitstrings b_1, b_2 by taking the polyomino corresponding to the concatenation of the two bitstrings $b_1 b_2$.

Note that the polyomino corresponding to a variable or clause can only mate with its complementary polyomino, it can not fit together with any polyomino corresponding to any other variable or clause or the complement thereof. Our construction uses as building blocks two more polyominoes: the *wildcard polyomino*, which is obtained as the polyomino corresponding to the bitstring 00001110000000...00000001110000 (i.e., $4\lceil \log(n+m) \rceil$ zeroes surrounded by 00001110000), and the *blocking polyomino*, which is the complementary polyomino for the wildcard. Note that the wildcard polyomino fits together with any clause or variable polyomino, while the blocking polyomino only fits together with the wildcard polyomino.

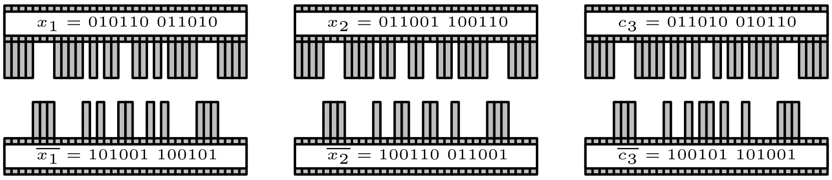

Fig. 1. Top: polyominoes corresponding to variables x_1, x_2 and clause c_3. Bottom: the complementary polyominoes, that mate with the polyominoes above them to form a $3 \times k$ square. Note that the polyominoes are depicted compressed horizontally.

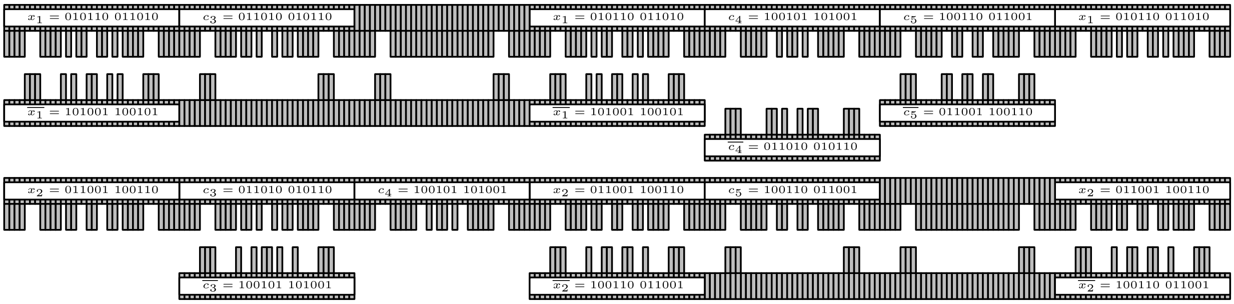

Fig. 2. Example of our reduction for the formula $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$. Top-to-bottom, left-to-right: formula encoding polyomino for x_1, variable-setting polyomino for x_1, clause-checking polyomino for c_4, clause checking-polyomino for c_5, formula-encoding polyomino for x_2, clause-checking polyomino for c_3, variable-setting polyomino for x_2. The polyominoes are arranged in a way that suggests the solution $x_1 = false$, $x_2 = true$.

Since each variable occurs as a literal at most three times, we can assume that it appears at most twice in positive form, and at most twice negated (if the variable occurs exclusively positively or negated we can simply remove the clauses that contain it to obtain an equivalent instance).

We are now ready to define the *formula-encoding polyominoes*. The construction will have n variable-encoding polyominoes, one for each variable x_i, and each consists of the concatenation of 7 polyominoes: we start with a polyomino corresponding to the bitstring of x_i. Next, for each time (at most two) x_i occurs positively in a clause, we take a polyomino corresponding to (the bitstring of) that clause. If x_i occurs only once in positive form, then we take (for padding) a copy of the blocking polyomino. Then, we take another copy of the polyomino for x_i. Next, we take the polyominoes corresponding to clauses in which x_i occurs negated. Again, we add the blocking polyomino if x_i only occurs negated once. Finally, we take another copy of the polyomino corresponding to x_i.

The *variable-setting polyomino* for x_i is the polyomino formed by concatenating, in the following order: (a) the complement polyomino for the variable, (b) 2 copies of the wildcard polyomino, (c) another copy of the complement polyomino.

The *clause-checking polyominoes* are simply the following: for each clause, we take a polyomino corresponding to the complement of its bitstring.

This completes the construction. An example of the construction is shown in Fig. 2. Note that if fixed or one-sided polyominoes are used, the formula-encoding ones are provided with the solid row of squares on top, and the remaining polyominoes are provided with the solid row on the bottom. We claim this set of polyominoes can be packed into a $3 \times 7n(22 + 4\lceil \log(n+m) \rceil)$ box if and only if the formula is satisfiable.

($\Rightarrow$). Suppose the polyominoes can be packed in a $3 \times 7n(22 + 4\lceil \log(n+m) \rceil)$ box. We first examine the placement of the formula-encoding polyominoes. Because each formula-encoding polyomino starts with a row of four ones, and the largest "gap" of zeroes occurring in one is of length three, they cannot overlap vertically; each formula-encoding polyomino must be fully to the right of the previous. Moreover, since the width of the target rectangle matches exactly the total width of the formula-encoding polyominoes, they must be placed back-to-back in some arbitrary permutation.

Consider the placement of a single complementary polyomino for a clause or variable. Because wherever two formula-encoding polyominoes touch back-to-back there are 8 consecutive rows in which 2 squares are already occupied, and the longest "gap" in a complementary polyomino is of length at most 5 (and at the left and right edges, there is a gap of length exactly 4, we see that the rows in which this polyomino are placed can contain only a single formula-encoding polyomino. This rules out any undesirable shifts: no complementary polyomino can overlap (vertically) more than one formula-encoding polyomino. Moreover, note that this same phenomenon forces the vertical alignment of polyominoes corresponding to variables or clauses in the formula-encoding polyominoes with the complementary polyominoes in variable-setting and clause-checking polyominoes.

Now, consider the placement of a variable-setting polyomino (for variable x_i). Since it starts with a complementary polyomino for x_i, and also ends with one x_i, it must be placed such that it only overlaps at most (and exactly) one formula-encoding polyomino, namely the one for x_i. It thus suffices to consider each formula-encoding polyomino in isolation. Note that then, there are only two possible placements for the variable-setting polyomino for variable x_i: either overlapping the first half of the formula-encoding polyomino, with the wildcard polyominoes used as building blocks in the variable-setting polyomino overlapping (and thus blocking) the polyominoes corresponding to clauses that are satisfied by setting x_i to *true*, or, overlapping the second half of the formula-encoding polyomino, overlapping (and thus blocking) the polyominoes corresponding to clauses that are satisfied by setting x_i to $false$.

Thus, the placement of the variable-setting polyominoes (unsurprisingly) corresponds to an assignment for the variables of the formula. It is easy to see that the clause-checking polyominoes can then be packed into the space left only if the assignment is satisfying: if the assignment does not satisfy some clause, then all the places where the respective clause-checking polyomino could fit are blocked by variable-setting polyominoes.

($\Leftarrow$). We can consider each formula-encoding polyomino in isolation. An assignment for the formula immediately tells us how to pack the variable-setting polyomino for x_i into the formula-encoding polyomino for x_i (namely: if x_i is true we place the variable-setting polyomino in the second half, otherwise, we place it in the first half of the formula-encoding polyomino). It is easy to see that if the assignment is satisfying, then for each clause-checking polyomino there is at least one possible placement inside a formula-encoding polyomino. For an example of how the pieces fit together for a satisfying assignment, see Fig. 2. □

Remark that our reduction leaves gaps inside the packing. If we consider the variant of the problem where total area of the pieces is equal to the area of the target shape, and thus the entire rectangle must be filled (Exact Polyomino Packing), the instance can be padded with several 1×1 polyominoes to make the total area of the pieces equal to the area of the target rectangle.

Corollary 1. *Unless the Exponential Time Hypothesis fails, there exists no $2^{o(n/\log n)}$-time algorithm for* Exact Polyomino Packing, *even if the target shape is a $3 \times n$ box, and the bounding box of each polyomino is of size $2 \times O(\log n)$.*

An interesting question is whether Corollary 1 still holds when the pieces are similarly sized, that is, each piece must have area $\Theta(\log n)$ (or even just $\Omega(\log n)$). An analog of Corollary 1 for exact packing of similarly-sized polyominoes can be achieved in the $4 \times n$ case:

Theorem 2. *Unless the Exponential Time Hypothesis fails, there exists no $2^{o(n/\log n)}$-time algorithm for* Exact Polyomino Packing, *even if the target shape is a $4 \times n$ box, the bounding box of each polyomino is of size $3 \times O(\log n)$ and the area of each polyomino is $\Theta(\log n)$.*

Proof. By a modification of the construction for the proof of Theorem 1. A corresponding (or complementary) polyomino can be *padded* by doubling the solid row, i.e., for a bitstring of length k we obtain a polyomino that fits inside a $3 \times k$ rectangle, with two solid (top) rows consisting of k squares each, and a (bottom) row with squares corresponding to the bitstring.

We make the following modifications:

- The clause-checking polyominoes are changed by padding them with an additional solid row, expanding their bounding boxes from $2 \times O(\log n)$ to $3 \times O(\log n)$.
- The variable-setting polyominoes are obtained by padding them with an additional row, and then removing any squares that corresponded to wildcard polyominoes in the original construction.
- The formula-encoding polyominoes remain unmodified.

Now, to ensure the target rectangle can be packed exactly, we add the following additional pieces:

(a) For every clause c_i consisting of p literals, we add $p-1$ (unpadded) complementary polyominoes.
(b) We add $2n-m$ copies of a $1 \times 22 + 4\lceil\log(n+m)\rceil$ rectangle polyomino.
(c) For every variable, we add one padded complementary polyomino.
(d) For every blocking polyomino occurring in the construction of the formula-encoding polyominoes, we add one (unpadded) wildcard polyomino.

Thus, for every p-literal clause c_i we have one padded complementary polyomino and $p-1$ unpadded complementary polyominoes. The padded complementary polyomino serves to verify that the clause is satisfied, while the unpadded ones can be packed into the gap left between a formula-encoding polyomino and a variable-setting polyomino, or can be combined with one of the rectangles (b) to fill the gap if the clause is satisfied by more than one of its literals. □

An interesting open question is to establish the exact complexity of Exact Polyomino Packing in the $3 \times n$ case with polyominoes of area $\Omega(\log n)$. It is easy to obtain a $2^{O(n/\log^2 n)}$-time lower bound by scaling up the construction of Corollary 1 in the horizontal axis by a factor $\log n$. Is a better algorithm or lower bound possible?

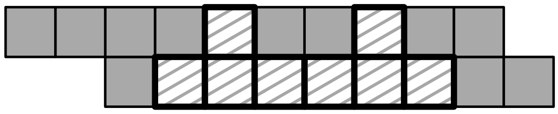

Fig. 3. Packing an arbitrary (non-degenerate) $2 \times k$ polyomino into a Y-monotone polyomino results in several pieces that are again Y-monotone.

Note that we can fix the position of the formula-encoding polyominoes in advance. The problem then reduces to packing variable-setting and clause-checking polyominoes into the shape left when subtracting the formula-encoding polyominoes from the $3 \times n$ rectangle, which fits inside a $2 \times n$ rectangle. Doing so we obtain the following corollary:

Corollary 2. *Unless the Exponential Time Hypothesis fails, there exists no $2^{o(n/\log n)}$-time algorithm for* Polyomino Packing *(resp.,* Exact Polyomino Packing*), even if the target shape fits inside a $2 \times n$ box, and the bounding box of each polyomino is of size $2 \times O(\log n)$ and each polyomino has area $\Theta(\log n)$ (resp., area $O(\log n)$).*

3. Algorithms

Our lower bound applies in a rather constrained case: even for packing polyominoes with a bounding box of size $2 \times O(\log n)$ into a rectangle of size $3 \times n$, there is no $2^{o(n/\log n)}$-time algorithm. As we will show later, a similar lower bound can not be established when the pieces are $1 \times k$ or $2 \times k$ rectangles (since the number of distinct such polyominoes is linear in their area rather than exponential). An interesting question, which we answer negatively, is whether a $2^{\Omega(n/\log n)}$-time lower bound can be obtained for packing polyominoes with a bounding box of size $2 \times O(\log n)$ into a rectangle of size $2 \times n$. Thus, the case for which we have derived our lower bound is essentially the most restrictive possible.

Note that, while solvable in strongly subexponential time, this problem is NP-complete, as can be seen by a simple reduction from 3-Partition: given a collection of integers $s_1, \ldots, s_{3k}$, take a $1 \times s_i$ rectangle for every integer s_i and one large "frame" polyomino consisting of a $2 \times (M + k + 1)$ rectangle into which k slots of size $1 \times M/k$ have been cut, where $M = \Sigma_{i=1}^{3k} s_i$.

We say that a polyomino is *Y-monotone* if every row consists of a number of contiguous squares, that is, there are no gaps.

Theorem 3. Polyomino Packing *for fixed, free or one-sided polyominoes can be solved in $2^{O(n^{3/4}\log n)}$ time if the target shape is a $2 \times n$ rectangle.*

Proof. First, consider a simple $O(2^a a^{O(1)})$-time dynamic programming algorithm that decides whether $m \leq a$ polyominoes $p_1, \ldots, p_m$ can be packed into a target polyomino of area a: for any subset S of (the squares of) the target polyomino (there are 2^a such subsets) and $i \in [1, m]$, let $B(S, i)$ be the proposition "the polyominoes $p_k, p_{i+1}, \ldots, p_m$ can be packed into S". $B(S, m)$ is simply the proposition that S is the same polyomino as p_m; if $B(S, i+1)$ is known for all S then $B(S', i)$ can be computed by trying all (polyominally many) placements of p_i within S'.

We are now ready to present our algorithm for $2 \times n$ polyomino packing. If we are dealing with free or one-sided polyominoes we first guess how many (if any) of the 1×2 polyominoes should be used in the vertical orientation, and how many in the horizontal orientation. This thus converts them to fixed 1×2 or 2×1 polyominoes, and only increases the running time of the algorithm by a factor n.

We augment the previously presented algorithm with the following observation: when the target polyomino is a $2 \times n$ rectangle, and if we process the polyominoes in a fixed order, with the polyominoes that are $1 \times k$ rectangles being processed last (thus after the 2×1 polyominoes and any other polyominoes), then the target shapes considered by the dynamic programming algorithm are always the disjoint union of several Y-monotone polyominoes (cf. Fig. 3). Such polyominoes can be described by 3 integers: one giving the number of squares in the bottom row, one giving the number of squares in the top row, and one giving the shift of the top row relative to the bottom row. Note that this observation crucially depends on processing the $1 \times k$ polyominoes last, since removing them from a $2 \times k$ polyomino does not necessarily result in a shape that is Y-monotone, however, if only $1 \times k$ polyominoes remain, we can ensure this requirement remains satisfied because we can consider the top and bottom row of each polyomino in the target shape separately.

If each of these integers is (in absolute value) at most $n^{1/4} - 1$ we call the resulting polyomino *small*, otherwise, the polyomino is *large*. We can use the following more efficient description of the target shape: for each polyomino in the shape that is small, we give the number of such polyominoes in the target shape and we simply list each large polyomino. Since there are at most $2n^{3/4}$ distinct small polyominoes,[2] giving the quantity for each leads to at most $(2n)^{2n^{3/4}} \leq 2^{2n^{3/4}(\log n+1)}$ cases. There are at most $2n^3$ distinct large polyominoes, but the target shape contains at most $2n^{3/4}$ of them (since each

[2] The top and bottom rows can consist of $0, 1, \ldots, n^{1/4} - 1$ squares, while the shift can be $-(n^{1/4} - 2)$ to $n^{1/4} - 2$.

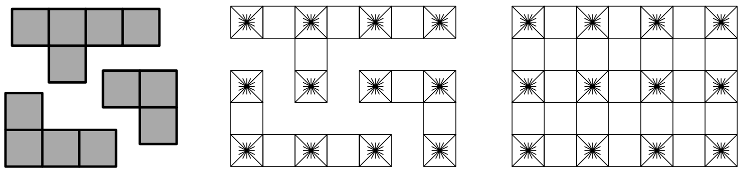

Fig. 4. Polyomino Packing problem (left) modelled as Subgraph Isomorphism from pattern (middle) into host graph (right).

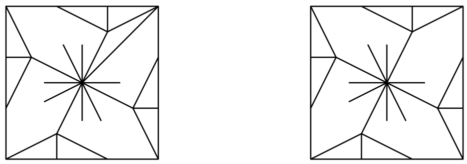

Fig. 5. Alternative constructions to use with fixed (left) or one-sided (right) polyominoes.

has area at least $n^{1/4}$), thus contributing $(2n^3)^{2n^{3/4}} \leq 2^{6n^{3/4}(\log n+1)}$ cases. Thus, if we identify equivalent target shapes, the dynamic programming algorithm needs to consider at most $2^{8n^{3/4}(\log n+1)}n = 2^{O(n^{3/4}\log n)}$ subsets of the target shape. $\square$

Note that this algorithm only works when the target shape is a $2 \times n$ rectangle. Corollary 2 shows that we should not expect a similar algorithm for packing polyominoes into an arbitrary target shape, even if that target shape fits in a $2 \times n$ box.

Finally, we show that our $2^{\Omega(n/\log n)}$-time lower bound is tight:

Theorem 4. Polyomino Packing *for free, fixed or one-sided polyominoes can be solved in* $2^{O(n/\log n)}$ *time if the target shape has area* n.

Proof. The problem can be modelled as Subgraph Isomorphism for an $O(n)$-vertex planar graph, for which a $2^{O(n/\log n)}$-time algorithm is known [3]. The construction is as follows: for every square in a polyomino, we take a cycle on four vertices, to which we add a fifth, universal vertex (which can be embedded in a planar embedding in the middle of this cycle). This fifth vertex is marked by adding a number of degree 1 vertices to it, to bring its degree up to (at least) 7. Each edge of this cycle is associated with an edge of the square in the polyomino. We make adjacent the endpoints of edges corresponding to adjacent edges in the polyomino. Both the host graph and the guest graph are constructed in this way, the host graph from the target shape (when viewed as a polyomino) and the guest graph from the set of input polyominoes (which will thus have one connected component corresponding to each separate polyomino that must be packed). An example for packing 3 polyominoes into a 3×4 rectangle is shown in Fig. 4. The special (degree 7) vertices must be mapped to other vertices that are also degree 7, and this means that the cycles corresponding to squares can only be mapped to cycles corresponding to other squares (and not to cycles created by making cycles adjacent since those vertices have degree less than 7).

This construction works for free polyominoes. To restrict to fixed or one-sided polyominoes, we can modify the construction slightly to make the structure used to represent a square asymmetric. For one-sided polyominoes, we create a structure that is rotationally symmetric but achiral. To this end, we subdivide each edge of the cycle twice and choose one of the two vertices created by this subdivision, add another vertex adjacent to this vertex, to its neighbours, and to the central vertex. For fixed polyominoes, we can add one additional edge (from the center to one of the vertices of the cycle) to also remove the rotational symmetry. These constructions are depicted in Fig. 5. $\square$

To make the paper self-contained and more instructional, we give a direct proof of the following weaker version of Theorem 4 — which illustrates in a simpler way the principles from [3].

Theorem 5. Polyomino Packing *for free, fixed or one-sided polyominoes can be solved in* $2^{O(n/\log n)}$ *time if the target shape is a rectangle of area* n.

Proof. If the rectangle is higher than it is wide, rotate it (and, if the polyominoes are fixed, the polyominoes as well) 90 degrees. Consider a scanline passing over the rectangle from left to right. At any given time, the scanline intersects at most $O(\sqrt{n})$ squares of the rectangle. We can specify how the intersection of the solution with the scanline looks by, for each square, specifying the polyomino (if any) that is placed there, along with its rotation and sidedness. Since the position of the polyomino can be fixed by specifying one of its squares, this gives at most $8n$ cases for each square, and, since the scanline intersects at most $\sqrt{n}$ squares, $2^{O(\sqrt{n}\log n)}$ cases total.

We furthermore need to specify which polyominoes have already been used in the solution (to the left of the scanline) and which ones still need to be packed. Similar to [3], a polyomino is *large* if it has area greater than $c\log_2 n$, and small otherwise. Since the number of polyominoes with area k is bounded by 4.65^k [8], the number of distinct small polyominoes it at most $4.65^{c \log_2 n}$. For $c \leq 0.22$, this is at most $\sqrt{n}$. We can specify the *quantity* of each small polyomino left with a single number from 0 to n, giving $(n+1)^{\sqrt{n}} = 2^{O(\sqrt{n}\log n)}$ cases. Meanwhile, the number of large polyominoes is at most $n/(c\log n)$, and thus there are $2^{O(n/\log n)}$ possible subsets of them.

The problem can now be solved by dynamic programming. For each position of the scanline, we have $2^{O(n/\log n)}$ subproblems: can a given subset of pieces ($2^{O(n/\log n)}$ cases) be packed entirely to the left of the scanline (with only the pieces intersecting the scanline possibly sticking out to the right of it), such that the intersection with the scanline looks as specified ($2^{O(\sqrt{n}\log n)}$ cases) (and, in the case of Exact Polyomino Packing, leaving no gaps)? For each such subproblem, we can find its answer by deleting the pieces whose leftmost square(s) intersect the scanline, and checking whether the instance thus obtained is compatible with some subproblem with the scanline moved one position to the left. □

There is an interesting contrast between these two algorithms. Whereas the strongly subexponential algorithm for the case of the $2 \times n$ rectangle works by considering the input polyominoes in a fixed order (so that we always know which subset we have used) and uses a bound on the number of subsets of the target shape that have to be considered, the algorithm for the general case works the opposite way around: it considers subsets of the target shape in a (more-or-less) fixed order (by the scanline approach) and bounds the number of possible subsets of the input polyominoes.

Note that our $2^{\Omega(n/\log n)}$-time lower bound exploits the fact that we can construct exponentially many polyominoes that fit inside a $2 \times O(\log n)$ rectangle. If we consider polyominoes with simpler shapes, we may be able to exploit this to obtain a faster algorithm. As an example, if the polyominoes are $a \times b$ rectangles, then the problem can be solved in strongly subexponential time:

Corollary 3. Polyomino Packing *can be solved in* $2^{O(\sqrt{n}\log n)}$ *time if the polyominoes are rectangular and the target shape is a rectangle with area n.*

Proof. Consider the algorithm presented in the proof of Theorem 5. The running time is dominated by the number of cases for tracking a subset of the polyominoes. If the polyominoes are rectangles, then note that the number of distinct rectangles of area at most n is also at most n. Call a polyomino *large* if it has area $\geq \sqrt{n}$ and *small* otherwise: there are at most $\sqrt{n}$ large polyominoes in the input, and thus at most $2^{\sqrt{n}}$ subsets of them. The number of distinct small polyominoes is at most $\sqrt{n}$, and thus specifying the quantity for each leads to at most $n^{\sqrt{n}} = 2^{\sqrt{n}\log n}$ cases. □

This problem is essentially equivalent to the Distributor's Pallet Loading Problem (DPLP) [9], which is a problem that has received considerable attention from an operations research perspective. The approach considered in [9] is somewhat similar to ours, in the sense that it also considers separators of the packing and uses dynamic programming to determine which subsets of the input boxes should be packed on which side of the separator. However, the analysis is mostly from the perspective of heuristics (e.g., trying to prune away unpromising solutions) rather than an exact complexity point of view.

4. Conclusions

In this paper, we have given a precise characterization of the complexity of (Exact) Polyomino Packing. For a set of polyominoes of total area n, the problem can be solved in $2^{O(n/\log n)}$ time. Even when restricted to the case where the pieces are of size $2 \times O(\log n)$ and they have to be packed into a $3 \times n$ rectangle or into a given shape which fits inside a $2 \times n$ rectangle, there is no faster (up to the base of the exponentiation) algorithm unless the Exponential Time Hypothesis fails. In contrast, in the case where the target shape is a $2 \times n$ rectangle, a strongly subexponential algorithm exists.

We conclude by listing several interesting open problems:

- Exact polyomino packing with excess pieces: we are given some target shape, and a set of polyominoes with total area possibly exceeding the target shape. Is it possible to use a subset of the polyominoes to build the target shape? Clearly this problem is at least as hard as (exact) polyomino packing; however, considering the set of pieces may be much larger than the target shape, it would be interesting to study this problem from a parameterized perspective (where the parameter k is the area of the target shape). The problem can be solved in $2^k n^{O(1)}$-time (by the simple dynamic programming algorithm of Section 3); is there a $2^{o(k)} n^{O(1)}$-time (or even a $2^{o(k)} 2^{o(n/\log n)}$-time) algorithm?
- What is the (exact) complexity of Exact Polyomino Packing when every piece has area $\Omega(\log n)$ or $\Theta(\log n)$ and the target polyomino is $3 \times n$? Our lower bound construction uses 1×1 polyominoes to fill the gaps in the packing. Requiring that each piece has area $\Omega(\log n)$ seems to limit the number of possible interactions between two pieces significantly if the target shape is a $3 \times n$ rectangle.
- We do not believe that our algorithm for packing polyominoes into a $2 \times n$ rectangle is tight. What is the exact complexity of this problem? This is closely related to the exact complexity of 3-Partition with the input given in unary, which (to our knowledge) is also an open problem.

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.

Acknowledgement

We thank an anonymous referee for the proof of Theorem 2, and for many useful comments and suggestions.

References

[1] R.A. Hearn, E.D. Demaine, Games, Puzzles, and Computation, CRC Press, 2009.
[2] R. Impagliazzo, R. Paturi, On the complexity of k-SAT, J. Comput. Syst. Sci. 62 (2) (2001) 367–375.
[3] H.L. Bodlaender, J. Nederlof, T.C. van der Zanden, Subexponential time algorithms for embedding H-minor free graphs, in: I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, D. Sangiorgi (Eds.), 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), in: Leibniz International Proceedings in Informatics (LIPIcs), vol. 55, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2016, pp. 9:1–9:14.
[4] E.D. Demaine, M.L. Demaine, Jigsaw puzzles, edge matching, and polyomino packing: connections and complexity, Graphs Comb. 23 (1) (2007) 195–208.
[5] M. Brand, Small polyomino packing, Inf. Process. Lett. 126 (2017) 30–34.
[6] H.L. Bodlaender, T.C. van der Zanden, Improved lower bounds for graph embedding problems, in: D. Fotakis, A. Pagourtzis, V.T. Paschos (Eds.), 10th International Conference on Algorithms and Complexity (CIAC 2017), in: LNCS, vol. 10236, Springer, 2017, pp. 92–103.
[7] R. Impagliazzo, R. Paturi, F. Zane, Which problems have strongly exponential complexity?, J. Comput. Syst. Sci. 63 (2001) 512–530.
[8] D.A. Klarner, R.L. Rivest, A procedure for improving the upper bound for the number of n-ominoes, Can. J. Math. 25 (3) (1973) 585–602.
[9] T.J. Hodgson, A combined approach to the pallet loading problem, AIIE Trans. 14 (3) (1982) 175–182.