On the Exact Complexity of Polyomino Packing

Hans L. Bodlaender
Department of Computer Science, Utrecht University, Utrecht, The Netherlands and
Department of Mathematics and Computer Science, Eindhoven University of Technology,
Eindhoven, The Netherlands
H.L.Bodlaender@uu.nl

Tom C. van der Zanden
Department of Computer Science, Utrecht University, Utrecht, The Netherlands
T.C.vanderZanden@uu.nl

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.

2012 ACM Subject Classification Theory of computation → Problems, reductions and com-
pleteness, Mathematics of computing → Combinatorial algorithms

Keywords and phrases polyomino packing, exact complexity, exponential time hypothesis

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. [5] 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 [6].

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.
**On the Exact Complexity of Polyomino Packing**

Polyomino Packing appears to behave similarly to Subgraph Isomorphism on planar graphs, which has exact complexity $2^{O(n/\log n)}$ [1] (i.e., there exist 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 $O(\log n)$. This gap was recently closed by Brand [3], 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 $2^{O(n/\log n)}$-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, and the bounding box of each polyomino is of size $2 \times \Theta(\log n)$.

**Proof.** A weaker version of the statement follows by a simple reduction from the Orthogonal Vector Crafting problem [2]. 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 [2].

We proceed by reduction from $n$-variable 3-SAT, which, unless the Exponential Time Hypothesis fails, does not admit a $2^{o(n)}$-time algorithm. By the Sparsification Lemma [7], we can assume that 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 $(-x_{i,1} \lor x_{i,2})(-x_{i,2} \lor x_{i,3}) \land \ldots \land (-x_{i,k-1} \lor x_{i,k}) \land (\neg x_{i,k} \lor x_{i,1})$. 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.

Figure 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.

We can concatenate two polyominoes corresponding to bitstrings $b_1, b_2$ by taking the polyomino corresponding to the concatenation of the two bitstrings $b_1b_2$. 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...0000001110000 (4 $\log (n + m)$) 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.

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 Figure 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 Figure 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 $1 \times 1$ polyominoes to make the total area of the pieces equal to the area of the target rectangle.

\[\textbf{Corollary 2.}\] 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 $3 \times n$ box, and the bounding box of each polyomino is of size $2 \times O(\log n)$.

This raises an interesting open problem: does Exact Polyomino Packing still admit a $2^{\Omega(n/\log n)}$-time lower bound when the pieces are similarly sized, that is, each piece must have area $\Theta(\log n)$ (or even just $\Omega(n)$). This seems to greatly limit the number of possible interactions between two polyomino pieces, since they cannot be combined in a way that creates small gaps.

Note that in the previous reduction 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:

\[\textbf{Corollary 3.}\] 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 $2 \times \Theta(\log n)$ (resp., $2 \times O(\log n)$).
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 $\text{NP}$-complete, as can be seen by a simple reduction from 3-Partition.

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

\textbf{Theorem 4.} \textbf{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.}

\textbf{Proof.} First, consider a simple $O(2^n n^{O(1)})$-time dynamic programming algorithm that decides whether $m$ polyominoes $p_1, \ldots, p_m$ can be packed into a target polyomino of area $n$: for any subset $S$ of (the squares of) the target polyomino (there are $2^n$ such subsets) and $1 \leq k \leq m$, let $B(S, k)$ be the proposition “the polyominoes $p_k, p_{k+1}, \ldots, p_m$ 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 $S'$.

If we are dealing with free or one-sided polyominoes we first guess how many (if any) of the $1 \times 2$ polyominoes should be used in the vertical orientation, and how many in the horizontal orientation. This thus converts them to fixed $1 \times 2$ or $2 \times 1$ 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 $2 \times 1$ 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 (c.f. Figure 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 at most $n^{1/4} - 1$ we call the resulting polyomino \textit{small}, otherwise, the polyomino is \textit{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 $n^{3/4}$ distinct small polyominoes, giving the quantity for each leads to at most $(2n)^{n^{3/4}} \leq 2^{n^{3/4}}(\log n + 1)$ cases. There are at most $n^3$ distinct large polyominoes, but the target shape contains at most $2n^{3/4}$ of them (since each has area at least $n^{1/3}$), thus contributing $(n^3)^2n^{3/4} \leq 2n^{3/4}(\log n + 1)$ cases. Thus, if we identify equivalent target shapes, the dynamic programming algorithm needs to consider at most $2n^{3/4}(\log n + 1)$ subproblems, and each subproblem can be handled in polynomial time.

Note that this algorithm only works when the target shape is a $2 \times n$ rectangle. Corollary 3 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 5.** 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 [1]. 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) 9. 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 $3 \times 4$ rectangle is shown in Figure 4. The special (degree 9) vertices must be mapped to other vertices that are also degree 9, 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 9).

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 identify 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 Figure 5.

To make the paper self-contained and more instructional, we give a direct proof of the
following weaker version of Theorem 5 — which illustrates in a simpler way the principles
from [1].

**Theorem 6.** Polygon 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 translation
with respect to the square. This gives at most $O(n^3)$ 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 [1], a
polyomino is *large* if it has area greater than $c \log n$, 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 $4.65^{c \log n}$. 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 Polygon 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, that is, polyominoes that are $a \times b$ rectangles, then the
problem can be solved in strongly subexponential time:

**Corollary 7.** Polygon 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 6. 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.

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 interested 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 $2^{kn^{O(1)}}$-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 $\Theta(\log n)$? Our lower bound construction uses $1 \times 1$ 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.

- We do not believe that our algorithm for packing polyominos 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.

References


On the Exact Complexity of Polyomino Packing


