Tiling with Squares and Packing Dominos in Polynomial Time

Aamand, Anders; Abrahamsen, Mikkel; Rasmussen, Peter M.R.; Ahle, Thomas D.

Published in:
ACM Transactions on Algorithms

DOI:
10.1145/3597932

Publication date:
2023

Document version
Publisher's PDF, also known as Version of record

Document license:
Unspecified

Citation for published version (APA):
https://doi.org/10.1145/3597932
Tiling with Squares and Packing Dominos in Polynomial Time

ANDERS AAMAND, Computer Science and Artificial Intelligence Lab, MIT, USA
MIKKEL ABRAHAMSEN and PETER M. R. RASMUSSEN, BARC, University of Copenhagen, Denmark
THOMAS D. AHLE, Meta, USA

A polyomino is a polygonal region with axis-parallel edges and corners of integral coordinates, which may have holes. In this paper, we consider planar tiling and packing problems with polyomino pieces and a polyomino container $P$. We give polynomial-time algorithms for deciding if $P$ can be tiled with $k \times k$ squares for any fixed $k$ which can be part of the input (that is, deciding if $P$ is the union of a set of non-overlapping $k \times k$ squares) and for packing $P$ with a maximum number of non-overlapping and axis-parallel $2 \times 1$ dominos, allowing rotations by $90^\circ$. As packing is more general than tiling, the latter algorithm can also be used to decide if $P$ can be tiled by $2 \times 1$ dominos.

These are classical problems with important applications in VLSI design, and the related problem of finding a maximum packing of $2 \times 2$ squares is known to be NP-hard [6]. For our three problems there are known pseudo-polynomial-time algorithms, that is, algorithms with running times polynomial in the area or perimeter of $P$. However, the standard, compact way to represent a polygon is by listing the coordinates of the corners in binary. We use this representation, and thus present the first polynomial-time algorithms for the problems. Concretely, we give a simple $O(n \log n)$-time algorithm for tiling with squares, where $n$ is the number of corners of $P$. We then give a more involved algorithm that reduces the problems of packing and tiling with dominos to finding a maximum and perfect matching in a graph with $O(n^3)$ vertices. This leads to algorithms with running times $O(n^3 \log^3 n \log \log n)$ and $O(n^3 \log^2 n \log \log n)$, respectively.

CCS Concepts: • Theory of computation → Design and analysis of algorithms; Computational geometry;

Additional Key Words and Phrases: Packing, tiling, polyominoes

ACM Reference format:
https://doi.org/10.1145/3597932
1 INTRODUCTION

A chessboard has been mutilated by removing two diagonally opposite corners, leaving 62 squares (see Figure 1). Philosopher Max Black asked in 1946 whether one can place 31 dominoes of size $1 \times 2$ so as to cover all of the remaining squares. Tiling problems of this sort are popular in recreational mathematics, such as the mathematical olympiads\footnote{See e.g., the “hook problem” of the International Mathematical Olympiad 2004.} and have been discussed by Golomb [17] and Gamow and Stern [15]. The mutilated chessboard and the dominoes are examples of the type of polygon called a polyomino, which is a polygonal region of the plane with axis-parallel edges and corners of integral coordinates. We allow polyominos to have holes.

From an algorithmic point of view, it is natural to ask whether a given (large) polyomino $P$ can be tiled by copies of another fixed (small) polyomino $Q$, which means that $P$ is the union of non-overlapping copies of $Q$ that may or may not be rotated by $90^\circ$ and $180^\circ$. As the answer is often a boring no, one can ask more generally for the largest number of copies of $Q$ that can be packed into the given container $P$ without overlapping. Algorithms answering this question (for various $Q$) turn out to have important applications in very large scale integration (VLSI) circuit technology. As a concrete example, Hochbaum and Maass [19] gave the following motivation for their development of a polynomial-time approximation scheme for packing $2 \times 2$ squares into a given polyomino $P$ (using the area representation of $P$, to be defined later).

"For example, 64K RAM chips, some of which may be defective, are available on a rectilinear grid placed on a silicon wafer. $2 \times 2$ arrays of such nondefective chips could be wired together to produce 256K RAM chips. In order to maximize yield, we want to pack a maximal number of such $2 \times 2$ arrays into the array of working chips on a wafer."

Although the mentioned amounts of memory are small compared to those of present day technology, the basic principles behind the production of computer memory are largely unchanged, and methods for circumventing defective cells of wafers (the cells are also known as dies in this context) is still an active area of research in semiconductor manufacturing [10, 12, 22, 25].

The most important result in tiling is perhaps the combinatorial group theory approach by Conway and Lagarias [11]. Their algorithmic technique is used to decide whether a given finite region consisting of cells in a regular lattice (triangular, square, or hexagonal) can be tiled by pieces drawn from a finite set of tile shapes. Thurston [32] gives a nice introduction to the technique and shows how it can be used to decide if a polyomino without holes can be tiled by dominos. The running time is $O(a \log a)$, where $a$ is the area of $P$. Pak, Sheffer, and Tassy [28] described an algorithm with running time $O(p \log p)$, where $p$ is the perimeter of $P$.

The problem of packing a maximum number of dominoes into a given polyomino $P$ was apparently first analyzed by Berman, Leighton, and Snyder [7] who observed that this problem can be reduced to finding a maximum matching of the incidence graph $G(P)$ of the cells in $P$: There is a vertex for each $1 \times 1$ cell in $P$, and two vertices are connected by an edge if the two cells share a geometrical edge. The graph $G(P)$ is bipartite, so the problem can be solved in $O(a^{3/2})$ time using the Hopcroft–Karp algorithm [20], where $a$ is the number of cells (i.e., the area of $P$).

On the flip-side, a number of hardness results have been obtained for simple tiling and packing problems. Beauquier, Nivat, Remila, and Robson [3] showed that if $P$ can have holes, the problem of deciding if $P$ can be tiled by translates of two rectangles $1 \times m$ and $k \times 1$ is NP-complete as soon as $\max\{m, k\} \geq 3$ and $\min\{m, k\} \geq 2$. Pak and Yang [29] showed that there exists a set of at most $10^6$ rectangles such that deciding whether a given hole-free polyomino can be tiled with translates from the set is NP-complete. Other generalizations have even turned out to be undecidable:
Berger [5] proved in 1966 that deciding whether pieces from a given finite set of polyominos can tile the plane is Turing-complete (interestingly, Wijshoff and van Leeuwen [34] and Beauquier and Nivat [2] gave algorithms for deciding whether a single polyomino tiles the plane). For packing, Fowler, Paterson, and Tanimoto [14] showed already in the early 80s that deciding whether a given number of $3 \times 3$ squares can be packed into a polyomino (with holes) is NP-complete, and the result was strengthened to $2 \times 2$ squares by Berman, Johnson, Leighton, Shor, and Snyder [6].

As it turns out, for all of the above results, it is assumed that the container $P$ is represented either as a list of the individual cells forming the interior of $P$ or as a list of the boundary cells. We shall call these representations the area representation and perimeter representation, respectively. The area and perimeter representations correspond to a unary rather than binary representation of integers and the running times of the existing algorithms are thus only pseudo-polynomial. It is much more efficient and compact to represent $P$ by the coordinates of the corners, where the coordinates are represented as binary numbers. This is the way one would usually represent polygons (with holes) in computational geometry: The corners are given in cyclic order as they appear on the boundary of $P$, one cycle for the outer boundary and one for each of the holes of $P$. We shall call such a representation a corner representation. With a corner representation, the area and perimeter can be exponential in the input size, so the known algorithms which rely on an area or perimeter representation to be polynomial, are in fact exponential when using this more efficient encoding of the input. Problems that are NP-complete in the area or perimeter representation are also NP-hard in the corner representation, but NP-membership does not necessarily follow. In our practical example of semiconductor manufacturing, the corner representation also seems to be the natural setting for the problem: Hopefully, there are only few defective cells to be avoided when grouping the chips, so the total number of corners of the usable region is much smaller than its area.

El-Khechen, Dulieu, Iacono, and Van Omme [13] showed that even using a corner representation for a polyomino $P$, the problem of deciding if $m$ squares of size $2 \times 2$ can be packed into $P$ is in NP. That was not clear before since the naive certificate specifies the placement of each of the $m$ squares, and so, would have exponential length. Beyond this, we know of no other work using the corner representation for polyomino tiling or packing problems.

Our contribution. While the complexity of the problem of packing $2 \times 2$ squares into a polyomino $P$ has thus been settled as NP-complete, the complexity of the tiling problem was left unsettled. Tiling and packing are closely connected in this area of geometry, but their complexities can be drastically different. Indeed, we show in Section 3 that it can be decided in $O(n \log n)$ time by a
surprisingly simple algorithm whether $P$ can be tiled by $k \times k$ squares for any fixed $k \in \mathbb{N}$ which can even be part of the input. Here, $n$ is the number of corners of $P$ and we assume throughout the paper that we can make basic operations (additions, subtractions, comparisons) on the coordinates in $O(1)$ time. We will discuss this choice of computational model shortly. With the area and perimeter representations, it is trivial to decide if $P$ can be tiled in polynomial time (see Section 3), but as noted above, using the corner representation, it is not even immediately obvious that the problem is in NP. We show in Section 3.5 that if $P$ does not have holes, deciding whether $P$ can be tiled can even be done in $O(n)$ time by a conceptually simple algorithm. This, however, relies on Chazelle’s algorithm [9], which is infamous for being complicated.

In Section 4, we provide and analyze a simple algorithm, which we denote simple-packer, that can decide if $m$ dominos (i.e., rectangles of size $1 \times 2$ that can be rotated $90^\circ$) can be packed in a given polyomino $P$. The algorithm works by truncating long edges of $P$, so that the resulting polyomino $P''$ has area $O(n^4)$. The graph $G(P'')$ induced by the unit square cells constituting $P''$ can likewise be constructed in time $O(n^3)$. We then use a multiple-sink multiple-source maximum flow algorithm as a black box [8, 16] to find a maximum matching in $G(P'')$, which results in a running time of $O(n^4 \frac{\log^2 n}{\log \log n})$. In order to decide if $P$ can be tiled with dominos, we can instead use a single-source shortest path algorithm [27], with which one can find perfect matchings in bipartite planar graphs [26], and we obtain a slightly better running time of $O(n^4 \frac{\log^2 n}{\log \log n})$. Although the truncation process of reducing the size to $O(n^3)$ is simple, the proof of correctness is nontrivial and requires some structural lemmas on domino packings.

In Section 5, we manage to reduce the domino packing and tiling problems to finding a maximum and perfect matching in a bipartite planar graph $G^*$ with $O(n^3)$ vertices, instead of $O(n^3)$ as for simple-packer. We denote this algorithm fast-packer. The actual graph $G^*$ can also be constructed in time $O(n^3)$. This reduction relies on the same structural results as are needed for simple-packer, but it is however quite a bit more complicated, and many techniques and technical lemmas are required to prove correctness and bound the size of $G^*$. We obtain running times of $O(n^3 \frac{\log^2 n}{\log \log n})$ and $O(n^3 \frac{\log^2 n}{\log \log n})$ for packing and tiling, respectively.

Table 1 summarises the known and new results.

**Choice of computational model.** We assume a unit-cost word RAM model where we can make basic operations (additions, subtractions, comparisons) on the corners in $O(1)$ time. In this model, we obtain linear and cubic running times in $n$ (up to logarithmic factors) for tiling with $2 \times 2$ squares and packing dominos. The word RAM model is a standard computational model and it simplifies our exhibition. However, one can argue that from a certain perspective, it is not the appropriate choice of model. Indeed, we are mainly concerned with instances where the corner coordinates are very large (e.g., exponential in $n$) because it is for these instances that the classic matching algorithms are only pseudo-polynomial. For such very large coordinates, it might be unrealistic to expect that we can perform the basic operations in $O(1)$ time. Suppose that $t$ is the...
time it takes to make one of the basic operations on corner coordinates. Then one can check that the time complexity of our square tiling algorithm will be \(O(nt \log n)\) and the domino packing algorithm will have complexity \(O(n^3 t + n^2 \log^3 n / \log \log n)\). For the domino packing algorithm, the first term is the time it takes to obtain the reduced graph \(G^*\) (of size \(O(n^3)\)) and the latter term comes from running the blackbox algorithm on \(G^*\). For \(t\) very large, this may no longer be polynomial in \(n\). However, if we need \(b\) bits to represent the corners of the polyomino (where \(b\) might be very large, e.g., polynomial in \(n\)), then the input size \(s\) is \(\Theta(bn)\). Moreover, dispensing with the word RAM model, we can at the very least perform the basic operations in \(t = O(b)\) time, so the time complexity will be \(O(s \log s)\) for the square tiling algorithm and \(O(s^3 + s^3 \log^3 s / \log \log s)\) for the domino packing algorithm. Thus, the running time is indeed polynomial in the input size.

Open problems. Many interesting questions remain for slightly more complex shapes than studied in this paper. For instance, polynomial-time algorithms are known for tiling polyominos with larger rectangles in the area representation [23, 31]. Are there also polynomial-time algorithms in the corner representation? For the problems that are NP-complete in the area representation [3, 21, 29], which are also contained in NP in the corner representation?

Another interesting problem is to design domino tiling and packing algorithms with better running times than our \(O(n^3)\) in the corner representation, e.g., near-linear time algorithms. It seems conceivable that the techniques of [28] can lead to such improvements for tiling simply connected polyominos with dominos. Specifically, it is shown that to decide tileability of a simply connected polyomino \(P\), it suffices to check a certain Lipschitz condition on a height function defined on the boundary \(\partial P\) of \(P\), and that (essentially) this check can be carried out by considering only \(O(p)\) pairs of boundary points, where \(p\) is the perimeter of \(P\). It is plausible that one could obtain a similar bound on the number of pairs to be checked in terms of \(n\), which would lead to a faster domino tiling algorithm for hole-free polyominos.

Let us finally mention that the known NP-hardness proof of packing \(2 \times 2\) squares in a polyomino relies heavily on the use of holes. It is a well-known open problem to determine the complexity of finding a maximum packing of \(2 \times 2\) squares in a simply-connected polyomino. The problem is conjectured to be polynomial-time solvable [1]. We note that [33] provides polynomial time algorithms for solving the problem for a large class of polyominos which includes staircases, pyramids, and Manhattan skyline polyominos.

1.1 Our Techniques

Tiling with \(k \times k\) squares. We sort the corners of the given polyomino \(P\) by the \(x\)-coordinates and use a vertical sweep-line \(\ell\) that sweeps over \(P\) from left to right. The intuition is that the algorithm keeps track of how the tiling looks in the region of \(P\) to the left of \(\ell\) if a tiling exists. As \(\ell\) sweeps over \(P\), we keep track of how the tiling pattern changes under \(\ell\). Each vertical edge of \(P\) that \(\ell\) sweeps over causes changes to the tiling, and we must update our data structure accordingly.

Packing dominos. The basic approach of both the simple-packer and the fast-packer algorithm is to reduce the packing problem in the polyomino \(P\) (with \(n\) corners) to a maximum matching problem in a graph \(G^*\) with only polynomially many vertices and edges. We prove that a maximum matching in \(G^*\) corresponds to a maximum packing of dominos in \(P\). The construction of \(G^*\) relies on some non-trivial structural results on domino packings.

The algorithm simple-packer first sorts the corners by \(x\)-coordinates and considers the corners in this order \(c_1, \ldots, c_n\). When \(x(c_{i+1}) - x(c_i) > 9n\), we move all the corners \(c_{i+1}, \ldots, c_n\) to the left by a distance of \(\approx x(c_{i+1}) - x(c_i) - 6n\), so that the new distance is \(\approx 6n\). We then do a similar truncation of vertical edges, and the resulting polyomino \(P''\) has area \(O(n^4)\). We define \(G^*\) as the
induced graph \( G^* := G(P'') \) and then compute a maximum matching \( M \) in \( G^* \) using a multiple-source multiple-sink maximum flow algorithm \([8, 16]\). The structural lemmas are used to ensure that the number of uncovered cells in maximum domino packings of the original polyomino \( P \) and the reduced \( P'' \) are the same, so it follows that a maximum domino packing in \( P \) has size 

\[
|M| + \frac{\text{area}(P) - \text{area}(P'')}{2}.
\]

In the case of the algorithm \texttt{fast-packer}, we reduce the packing problem to finding a maximum matching in a bipartite planar graph \( G^* \) with \( O(n^3) \) vertices. The number of dominos in a maximum packing in the original polyomino \( P \) is then \( |M| + \frac{\text{area}(P) - |V(G^*)|}{2} \), where \( M \) is a maximum matching in \( G^* \) and \( V(G^*) \) is the vertex set of \( G^* \). The construction of \( G^* \) requires many techniques and technical lemmas regarding the particular way we define intermediate polyominos and graphs that are used to eventually arrive at \( G^* \). The process consists of five steps, and they are illustrated in Figures 13–15 and described informally below.

In step 1, we find the maximum subpolyomino \( P_1 \subseteq P \) such that all corners of \( P_1 \) have even coordinates.

In step 2, we use a hole-elimination technique: By carving channels in \( P_1 \) from the holes to the boundary, we obtain a hole-free subpolyomino \( P_2 \subseteq P_1 \). The particular way we choose the channels is important in order to ensure that the final graph \( G^* \) has size only \( O(n^3) \).

In step 3, we apply a technique of reducing \( P \) by removing everything far from the boundary of \( P_2 \): We consider the subpolyomino \( Q \subseteq P_2 \) of all cells with at least some distance \( \Omega(n) \) to the boundary of \( P_2 \), and then we define \( P_3 := P \setminus Q \) (note that \( Q \) is removed from \( P \) and not from \( P_2 \)). The main insight is that any packing of dominos in \( P_3 \) can be extended to a packing of all of \( P \) that, restricted to \( Q \), is a \textit{tiling}. For this to hold, it turns out to be important that \( P_2 \) has no holes.

A crucial step is to prove that every cell in the polyomino \( P_3 \) has distance \( O(n) \) to the boundary of \( P_3 \) and that \( P_3 \) has \( O(n) \) corners. There may, however, still be an exponential number of cells in \( P_3 \) due to long \textit{pipes} (corridors), and these are identified in step 4.

In step 5, we contract these long pipes. The contraction is not carried out geometrically, but in the incidence graph \( G_3 := G(P_3) \) of the cells of \( P_3 \), by contracting long horizontal and vertical paths to single edges, and the resulting graph is \( G^* \).

All vertices of \( G^* \) correspond to cells of \( P_3 \) with distance at most \( O(n) \) from a corner of \( P_3 \), and since \( P_3 \) has \( O(n) \) corners, we get that \( G^* \) has size \( O(n^3) \). We then proceed by computing a maximum matching in \( G^* \) as described above.

2 PRELIMINARIES

For a subset \( D \subseteq \mathbb{R}^2 \), we use \( \overline{D} \) to denote the closure of \( D \), \( \text{Int} D \) to denote the interior of \( D \), and \( \partial D \) to denote the boundary of \( D \). We further let \( D^c := \mathbb{R}^2 \setminus \overline{D} \). We define a cell to be a \( 1 \times 1 \) square of the form \([i, i + 1] \times [j, j + 1], i, j \in \mathbb{Z} \). A subset \( P \subseteq \mathbb{R}^2 \) is called a polyomino if it is a finite union of cells. For a polyomino \( P \), we define \( G(P) \) to be the graph which has the cells in \( P \) as vertices and an edge between two cells if they share a (geometric) edge. We say that \( P \) is connected if \( G(P) \) is a connected graph. Figure 2(a) illustrates a connected polyomino. For a simple closed curve \( \gamma \subseteq \mathbb{R}^2 \), we denote by \( \text{Int} \gamma \) the interior of \( \gamma \). An alternative way to represent a connected polyomino is by a sequence of simple closed curves \( (\gamma_0, \gamma_1, \ldots, \gamma_h) \) such that (1) each of the curves follows the horizontal and vertical lines of the integral grid \( \mathbb{Z}^2 \), (2) for each \( i \in \{1, \ldots, h\} \), \( \text{Int} \gamma_i \subseteq \text{Int} \gamma_0 \), (3) for each distinct \( i, j \in \{0, \ldots, h\} \), \( \text{Int} \gamma_i \cap \text{Int} \gamma_j = \emptyset \), and (4) for distinct \( i, j \in \{0, \ldots, h\} \), \( \gamma_i \cap \gamma_j \subseteq \mathbb{Z}^2 \). For a connected polyomino \( P \), there exists a unique such sequence (up to permutations of \( \gamma_1, \ldots, \gamma_h \)) with \( P = \text{Int} \gamma_0 \setminus (\bigcup_{i=1}^h \text{Int} \gamma_i) \). It is standard to reduce our tiling and packing problems to corresponding tiling and packing problems for connected polyominos, so for simplicity we will assume that the input polyominos to our algorithms are connected. The \textit{corners} of a polyomino \( P \)
(specified by a sequence \((y_0, y_1, \ldots, y_h)\), are the corners of the curves \(y_0, \ldots, y_h\). We assume that an input polyomino with \(n\) corners is represented using \(O(n)\) words of memory by describing the corners of each of the curves \(y_0, \ldots, y_h\) in cyclic order.

Finally, we define packings and tilings. Let \(P\) be a polyomino and \(Q\) be a set of polyominos. A packing of \(P\) with \(Q\) is a set of polyominos \(\{Q_1, \ldots, Q_k\}\) such that (1) for each \(1 \leq i \leq k\) there exists \(a \in \mathbb{Z}^2\) and \(Q \in Q\) with \(Q_i = \{a + q | q \in Q\}\) and (2) \(\text{Int} Q_i \cap \text{Int} Q_j = \emptyset\) for \(1 \leq i < j \leq k\). If \(\bigcup_{i=1}^k Q_i = P\), we say that it is a tiling. When \(Q = \{[0, 1] \times [0, 2], [0, 2] \times [0, 1]\}\) we refer to these packings and tilings as domino packings and domino tilings.

In this paper we will exclusively work with the \(L_\infty\)-norm when measuring distances. For two points \(a, b \in \mathbb{R}^2\) we define \(\text{dist}(a, b) = ||a - b||_\infty\). For two subsets \(A, B \subseteq \mathbb{R}^2\) we define

\[
\text{dist}(A, B) = \inf_{(a, b) \in A \times B} \text{dist}(a, b).
\]

In our analysis, \(A\) and \(B\) will always be closed and bounded (they will in fact be polyominos), and then the inf can be replaced by a min. For a point \(x \in \mathbb{R}^2\) and a subset \(A \subset \mathbb{R}^2\), we will use \(\text{dist}(x, A)\) as a shorthand notation for \(\text{dist}((x), A)\). Finally, we need the notion of the offset \(B(A, r)\) of a set \(A \subseteq \mathbb{R}^2\) by a value \(r \in \mathbb{R}\). If \(r \geq 0\), we define

\[
B(A, r) := \left\{x \in \mathbb{R}^2 \mid \text{dist}(x, A) \leq r\right\},
\]

and otherwise, we define \(B(A, r)\) to be the closure of \(B(A^c, -r)^c\). Note that if \(r \geq 0\), we have \(A \subseteq B(A, r)\) and otherwise, we have \(B(A, r) \subseteq A\).

Note that a domino packing of \(P\) naturally corresponds to a matching of \(G(P)\) and we will often take this viewpoint. We therefore require some basic matching terminology and a result on how to extend matchings. Let \(G\) be a graph and \(M\) a matching of \(G\). A path \((v_1, \ldots, v_{2k})\) of \(G\) is said to be an augmenting path if \(v_1\) and \(v_{2k}\) are unmatched in \(M\) and for each \(1 \leq i \leq k - 1\), \(v_{2i}\) and 
\(v_{2i+1}\) are matched to each other in \(M\). Modifying \(M\) restricted to \(\{v_1, \ldots, v_{2k}\}\) by instead matching \((v_{2i-1}, v_{2i})\) for \(1 \leq i \leq k\), we obtain a larger matching which now includes the two vertices \(v_1\) and \(v_{2k}\). See Figure 2(b) for an illustration in the context of domino packings. We require the following basic result by Berge which guarantees that any non-maximum matching of \(G\) can always be extended to a larger matching using an augmenting path as above.

**Lemma 2.1 (Berge [4])**. Let \(G\) be a graph and \(M\) a matching of \(G\) which is not maximum. Then there exists an augmenting path between two unmatched vertices \(G\).

### 3 TILING WITH SQUARES

#### 3.1 Naive Algorithm

The naive algorithm to decide if \(P\) can be tiled with \(k \times k\) tiles works as follows. Consider any convex corner \(c\) of \(P\). A \(k \times k\) square \(S\) must be placed with a corner at \(c\). If \(S\) is not contained in \(P\), we conclude that \(P\) cannot be tiled with \(k \times k\) squares. Otherwise, we recursively check if \(P \setminus S\)
Fig. 3. Two instances that cannot be tiled. Left: The edge $e_2$ splits the only interval in $I$ into two smaller intervals. Then $e_3$ introduces a new interval with a different parity than the existing two. The edge $e_4$ makes the algorithm conclude that $P$ cannot be tiled since $e_4$ overlaps an interval with the wrong parity. Right: The edges $e_3$ and $e_4$ introduce new intervals that are merged with the existing one. Edge $e_6$ introduces an interval which is merged with the existing interval and the result has odd length, so the algorithm concludes that $P$ cannot be tiled.

can be tiled. This algorithm runs in time polynomial in the area of $P$ and also shows that if $P$ can be tiled, there is a unique way to do it.

### 3.2 Sweep-line Algorithm

For the ease of presentation, we focus on the case of deciding tileability using $2 \times 2$ squares. It is straightforward to adapt the algorithm to decide tileability by $k \times k$ squares for any fixed $k \in \mathbb{N}$, as explained in the end of this section.

Our algorithm for deciding if a given polyomino $P$ can be tiled with $2 \times 2$ squares uses a vertical sweep line that sweeps over $P$ from left to right. The intuition is that the algorithm keeps track of how the tiling looks in the region of $P$ to the left of $\ell$ if a tiling exists. As $\ell$ sweeps over $P$, we keep track of how the tiling pattern changes under $\ell$. Each vertical edge of $P$ that $\ell$ sweeps over causes changes to the tiling, and we must update our data structures accordingly.

Recall that if $P$ is tileable, then the tiling is unique. We define $T(P) \subseteq P$ to be the union of the boundaries of the tiles in the tiling of $P$, i.e., such that $P \setminus T(P)$ is a set of open $2 \times 2$ squares. If $P$ is not tileable, we define $T(P) := \perp$.

Consider the situation where the sweep line is some vertical line $\ell$ with integral $x$-coordinate $x(\ell)$. The algorithm stores a set $I$ of pairwise interior-disjoint closed intervals $I = \{I_1, \ldots, I_m\} \subseteq \mathbb{R}$, ordered from bottom to top. Each interval $I_i$ has endpoints at integers and represents the segment $I'_i := \{x(\ell)\} \times I_i$ on $\ell$. In the simple case that no vertical edge of $P$ has $x$-coordinate $x(\ell)$ (so that no change to the set $P \cap \ell$ happens at this point), the intervals $I$ together represent the part of $\ell$ in $P$, i.e., we have $P \cap \ell = \bigcup_{i \in [m]} I'_i$. If one or more vertical edges of $P$ have $x$-coordinate $x(\ell)$, then $P \cap \ell$ changes at this point and the intervals $I$ must be updated accordingly.

For each interval $I_i$, we store a parity $p(I_i) \in \{0, 1\}$, which encodes how the tiling must be at $I'_i$ if $P$ is tileable. For a tileable polyomino $P$, these parities $p(I_i)$ will satisfy the following parity invariant; see also Figure 3.

- If $p(I_i)$ and $x(\ell)$ have the same parity, then $I'_i \subseteq T(P)$, i.e., $I'_i$ follows the boundaries of some tiles and does not pass through the middle of any tile.
- Otherwise, $I'_i \cap T(P)$ consists of isolated points, i.e., $I'_i$ passes through the middle of some of the tiles and does not follow the boundary of any tile.
Let \( I = \emptyset \) be the vertical edges of \( P \) in sorted order. Initialize \( I = \emptyset \).

**Algorithm 1:** Polynomial Time Square Tiling

```plaintext
for j = 1, ..., k do
    Let \([y_0, y_1]\) be the interval of \( y \)-coordinates of \( e_j \).
    if the interior of \( P \) is to the left of \( e_j \)
        for each \( I_i \in I \) that overlaps \([y_0, y_1]\) do
            if \( I_i \) and \( x(e_j) \) have different parity
                return "no tiling"
            Remove \( I_i \) from \( I \), let \( J = I_i \setminus (y_0, y_1) \), and if \( J \neq \emptyset \), add the interval(s) in \( J \) to \( I \).
        else
            Make a new interval \( I := [y_0, y_1] \) with the parity \( p(I) := x(e_j) \mod 2 \) and add \( I \) to \( I \).
            if \( I \) has one or two true neighbors in \( I \) that also have the same parity as \( I \)
                Merge those intervals in \( I \).
            if \( j < k \) and \( x(e_{j+1}) > x(e_j) \) and some \( I_i \in I \) has odd length
                return "no tiling"
    return "tileable"
```

We say that two neighboring intervals \( I_i, I_{i+1} \) of \( I \) are true neighbors if \( I_i \) and \( I_{i+1} \) share an endpoint. In addition to the parity invariant, we require \( I \) to satisfy the following neighbor invariant: Any pair of true neighbors of \( I \) have different parity.

The pseudocode of the algorithm is shown in Algorithm 1. Initially, we sort all vertical edges after their \( x \)-coordinates and break ties arbitrarily. We then run through the edges in this order. Each edge makes a change to the set \( P \cap \ell \), and we need to update the intervals \( I \) accordingly so that the parity and the neighbor invariants are satisfied after each edge has been handled. Figure 3 shows examples of the two cases where the algorithm concludes that there is no tiling.

Consider the event that the sweep line \( \ell \) reaches a vertical edge \( e_j = \{x\} \times [y_0, y_1] \). If the interior of \( P \) is to the left of \( e_j \), then \( P \cap \ell \) shrinks. Each interval \( I_i \in I \) that overlaps \([y_0, y_1]\) must then also shrink, be split into two, or disappear from \( I \). This is handled by the for-loop at line 5. If the parity of one of these intervals \( I_i \) does not agree with the parity of \( e_j \), we get from the parity invariant that \( P \) cannot be tiled, and hence the algorithm returns "no tiling" at line 7.

If on the other hand the interior of \( P \) is to the right of \( e_j \), then \( P \cap \ell \) expands and a new interval \( I \) must be added to \( I \). This is handled by the else-part at line 9. The new interval \( I \) may have one or two true neighbors in \( I \). If one or two such neighbors also have the same parity as \( I \), we merge these intervals into one interval of \( I \). This ensures that the neighbor invariant is satisfied after \( e_j \) has been handled.

In line 13, we consider the case that we finished handling all vertical edges at some specific \( x \)-coordinate so that the sweep line will move to the right in order to handle the next edge \( e_{j+1} \) in the next iteration. If there is an interval \( I_i \) of odd length in \( I \), it follows from the parity invariant together with the neighbor invariant that \( P \) cannot be tiled, so the algorithm returns "no tiling" at line 14.

The above explanation of the algorithm argues that if the invariants hold before edge \( e_j \) is handled, they also hold after. It remains to argue that they also hold before the next edge \( e_{j+1} \) is handled in the case that the sweep line \( \ell \) jumps to the right in order to sweep over \( e_{j+1} \). In the open strip between the vertical lines containing \( e_j \) and \( e_{j+1} \), there are no vertical segments of \( P \). Hence, the pattern of the tiling \( T(P) \) must continue as described by the parities \( p(I_i) \) in between the edges \( e_j \) and \( e_{j+1} \), so the parity invariant also holds before \( e_{j+1} \) is handled.

We already argued that if the algorithm returns “no tiling”, then $P$ is not tileable. Suppose on the other hand that the algorithm returns "tileable". In order to prove that $P$ can then be tiled, we define for each $j \in [k]$ a polyomino $P_j \subseteq P$. We consider the situation where the sweep line $\ell$ contains $e_j$ and $e_j$ has just been handled by the algorithm. We then define $P_j$ to be the union of

- the part of $P$ to the left of $\ell$, and
- the rectangle $[x(\ell), x(\ell) + 1] \times I_i$ for each $I_i \in \mathcal{I}$ with a different parity than $x(\ell)$.

We first see that for each $j \in [k]$, we have $P_j \subseteq P$. To this end, we just have to check that the rectangles $[x(\ell), x(\ell) + 1] \times I_i$ are in $P$. If one such interval was not in $P$, there would be an edge of $P$ overlapping the segment $\{x(\ell)\} \times I_i$. Since $I_i$ has a different parity than $x(\ell)$, this would make the algorithm report "no tiling" at line 7, contrary to our assumption.

We now prove by induction on $j$ that each $P_j$ can be tiled. Since $P = P_k$, this is sufficient. Along the way, we will also establish that $P_1 \subseteq P_2 \subseteq \cdots \subseteq P_k$. When $j = 1$, we see that $P_j$ is empty, so the statement is trivial. Suppose now that $P_j$ can be tiled and consider $P_{j+1}$. Note that if $x(e_j) = x(e_{j+1})$, so that $\ell$ does not move, then $P_j = P_{j+1}$, since all intervals that are created or modified when handling $e_{j+1}$ have the same parity as $x(\ell)$, so in this case, $P_{j+1}$ is tileable because $P_j$ is.

Consider now the case $x(e_j) < x(e_{j+1})$. Note that as $P_j \subseteq P$ and $P_j$ is to the left of the vertical line $\ell$ with first coordinate $x(e_j) + 1$, we have $P_j \subseteq P_{j+1}$. We now consider the set $\overline{P_{j+1}} \setminus P_j$ and argue that it is tileable; see Figure 4. Let $I_1, \ldots, I_m$ be the intervals in $\mathcal{I}$ after $e_j$ was handled. For each $I_i$, we add a rectangle $X \times I_i$ to $P_j$ in order to obtain $P_{j+1}$, where $X \subseteq \mathbb{R}$ is an interval with left endpoint $x(e_j)$ or $x(e_j) + 1$ and right endpoint $x(e_{j+1})$ or $x(e_{j+1}) + 1$, and by the definition of $P_j$ and $P_{j+1}$, it follows that $X$ has even length. Since each $I_i$ also has even length (otherwise, the algorithm would have returned "no tiling" at line 14 when $e_j$ was handled), the difference $\overline{P_{j+1}} \setminus P_j$ is a union of rectangles with even edge lengths, so $P_{j+1}$ is tileable since $P_j$ is.

### 3.3 Runtime Analysis

Assuming that we can compare two coordinates in $O(1)$ time, we sort the vertical edges by their $x$-coordinates in $O(n \log n)$ time. Since the intervals of $\mathcal{I}$ are pairwise interior-disjoint, we can implement $\mathcal{I}$ as a balanced binary search tree, where each leaf stores an interval $I_i$. 

---

**Fig. 4.** The polyomino $P_j$ is the part of $P$ to the left of the line $x = x(e_j)$ (this part of $P_j$ is not shown) plus the grey rectangles along the line. Here, the difference $x(e_{j+1}) - x(e_j)$ is odd. The difference $\overline{P_{j+1}} \setminus P_j$ has been tiled with green $2 \times 2$ squares.
Fig. 5. Left: The linear-time algorithm for deciding if a hole-free polyomino $P$ can be tiled with $2 \times 2$ squares partitions $P$ into rectangles, constructs the dual tree $T$, and tiles from the leafs of $T$ towards the root. In this example, the algorithm decides that $P$ cannot be tiled when the root $r$ is reached, as the tiling patterns of the edges shared with the children of $r$ do not match. Right: A single rectangle with three children and a parent. The tiling patterns of the segments shared with the children are given. The algorithm checks that the patterns match along the vertical edges and constructs the pattern shared with the parent. In this case, the algorithm decides that there is no tiling, since the pattern shared with the parent has intervals of odd length.

We now argue that each vertical edge $e_j$, with $y$-coordinates $[y_0, y_1]$, takes only $O(\log n)$ time to handle, since we need to make only $O(1)$ updates to $I$. If the interior of $P$ is to the left of $e_j$, then $[y_0, y_1] \subseteq \bigcup_{i \in [m]} I_i$. It then follows from the neighbor invariant that if $[y_0, y_1]$ overlaps more than one interval $I_i$, then the algorithm will return “no tiling”. We therefore do at most $O(1)$ updates to $I$, so it takes $O(\log n)$ time to handle $e_j$.

On the other hand, if the interior of $P$ is to the right of $e_j$, we need to insert a new interval into $I$ and possibly merge it with one or two neighbors in $I$, so this also amounts to $O(1)$ changes to $I$.

At line 13, we need to check the $O(1)$ intervals that were added or changed due to the edge $e_j$, so this can be done in $O(1)$ time. Hence, the algorithm has runtime $O(n \log n)$.

### 3.4 Adaptation to $k \times k$ Squares

In order to adapt the algorithm to $k \times k$ squares, we need to compare coordinates modulo $k$ instead of modulo 2. Specifically, each interval in $I$ stores a number $p(I) \in \{0, 1, \ldots, k - 1\}$, which is set to $x(e_j) \mod k$ at line 10. We fail at line 6 if $x(e_j) \mod k \neq p(I)$ and at line 13 if some $I_i$ has a length not divisible by $k$. At line 12, we merge $I$ with the true neighbours that have the same $p$-value. With these modifications, all arguments carry over to the case of $k \times k$ squares with a similar running time of $O(n \log n)$ which does not depend on $k$.

### 3.5 Linear-time Algorithm for Polyominoes without Holes

If the container $P$ does not have any holes, we can decide if $P$ can be tiled with $2 \times 2$ squares in $O(n)$ time in the following way. We first partition $P$ into rectangles by computing a trapezoidation of $P$. Since $P$ is a polyomino, this corresponds to shooting a vertical ray from each reflex corner into the interior of $P$ until it hits the boundary. The trapezoids will be the maximal rectangles contained in $P$ with the horizontal edges contained in the boundary $\partial P$. This decomposition of $P$ can be constructed in $O(n)$ time using Chazelle’s algorithm [9]. The rectangles induce a tree $T$ since $P$ does not have holes; see Figure 5. We choose an arbitrary root $r$ in $T$ and, simply speaking, tile from the leafs and towards $r$, as described in the following.

Assuming that a tiling exists, we will represent the tiling pattern along the vertical edges of each rectangle using linked lists. Each element in a list stores an interval of $y$-coordinates and one bit indicating whether the tiling has even or odd parity in that interval, i.e., whether the vertical
edges of the tiles have even or odd $x$-coordinates. When handling a given rectangle $s$ of $T$, we already know the pattern of the tiling along the parts of the edges that $s$ shares with the children of $s$. The parts of the edges that $s$ shares with $\partial P$ are given by their $x$-coordinate and the interval spanned by the $y$-coordinates. If $s$ is not the root, one of the vertical edges shares a segment with the parent of $s$, but except from this, the tiling pattern along both vertical edges is fixed and can be constructed in $O(\delta(s))$ time, where $\delta(s)$ is the degree of $s$ in $T$.

In the intervals where the tiling patterns are fixed along both the left and right vertical edge of $s$, the patterns must match, and each interval of $y$-coordinates in the patterns must have even length. The pattern of the segment shared with the parent of $s$ is inherited from the pattern of the opposite edge. The time used to check that the patterns match and construct the pattern for the parent is proportional to the number of intervals that cancel out, which can be accounted for in the rectangles where each interval first appeared. Hence, the overall work is $O(n)$. This algorithm likewise generalizes to the problem of tiling with $k \times k$ squares.

4 SIMPLE DOMINO PACKING ALGORITHM

In this section we will present our polynomial-time algorithm, simple-packer, for finding the maximum number of $1 \times 2$ dominos that can be packed in a polyomino $P$. We assume that the dominos must be placed with axis-parallel edges, but they can be rotated by $90^\circ$. In any such packing, we can assume the pieces to have integral coordinates: if they do not, we can translate the pieces as far down and to the left as possible, and the corners will arrive at positions with integral coordinates. We first describe a naive algorithm which runs in polynomial time in the area of the polyomino.

4.1 Naive Algorithm

The naive algorithm considers the graph $G(P) = (V, E)$ where $V$ is the set of cells of $P$ and $e = (u, v) \in E$ if and only if the two cells $u$ and $v$ have a (geometric) edge in common. The maximum number of $1 \times 2$ dominos that can be packed in $P$ is exactly the size of a maximum matching of $G$ and it is well known that such a maximum matching can be found in polynomial time in $|V|$, i.e., in the area of $P$.

4.2 Simple Polynomial-time Algorithm

Our polynomial-time algorithm, simple-packer, first sorts the corners of $P$ by $x$-coordinates from left to right and considers the corners in this order $c_1, \ldots, c_n$. When $x(c_{i+1}) - x(c_i) > 9n$, we move all the corners $c_{i+1}, \ldots, c_n$ to the left by a distance of $2\left\lfloor \frac{x(c_{i+1}) - x(c_i)}{2} \right\rfloor - 6n$. We call this operation a contraction. The result after all of the contractions is a polyomino $P''$ with the parities of the $x$-coordinates unchanged and with the difference between the $x$-coordinates of any two consecutive corners at most $9n$. We then consider the corners in order according to $y$-coordinates and do a similar truncation of the long vertical edges. We have now reduced the container $P$ to an orthogonal polygon $P'''$ of area at most $O(n^4)$, since the span of the $x$-coordinates is $O(n^2)$, as is the span of the $y$-coordinates. We proceed by finding maximum or perfect matchings in $G(P''')$, as described in the introduction. To find a tiling or maximum packing in $P$ from one in $G(P''')$, we undo the contractions of $P$ and insert extra dominos. Each $x$-contraction was performed by moving the corners $c_{i+1}, \ldots, c_n$ an even distance to the left. This means that when undoing this contraction, we can simply insert extra horizontal dominos. Likewise, for undoing the $y$-contractions, we insert vertical dominos to complete the packing or tiling.

For some containers $P$, the graph $G(P''')$ really has $\Omega(n^4)$ vertices, so simple-packer is indeed slower than fast-packer. For instance when the boundary of $P$ consists of four "staircases", each consisting of $n/4$ vertices, where each step has width and height $n$; see Figure 6 (left). Here,
Fig. 6. Left: A polyomino with area \( \Omega(n^4) \) that \textit{simple-packer} will not reduce. Right: If we truncate edges so that consecutive \( x \)-coordinates have difference either 1 or 2 (keeping the parities invariant), then there may be more uncovered cells in a maximum packing of the reduced instance than in the original.

Fig. 7. A pipe \( Q \) of width \( k \) and length \( \ell \).

\textit{fast-packer} will remove most of the interior, leaving a layer of cells of thickness \( O(n) \) around the boundary, but \textit{simple-packer} will not make any contractions.

One might be tempted to think that we can even truncate the edges so that the difference between consecutive \( x \)- and \( y \)-coordinates is either 1 or 2, keeping the parity of all coordinates. However, this does not work, as seen in Figure 6 (right). Two dominos can be packed in the reduced container \( P' \), and the reduction decreases the area by eight cells, so the formula would give that the original container \( P \) has room for six dominos, but there is actually room for seven.

4.3 Structural Results on Polyominos and Domino Packings

Building up to our structural results on domino packings, we require a few definitions and simple lemmas. We first introduce the notion of a \textit{pipe} (see Figure 7) and \textit{consistent parity}.

\textbf{Definition 1.} Let \( P \) and \( Q \) be polyominos with \( Q \subseteq P \). We say that \( Q \) is a \textit{pipe} of \( P \) if \( Q \) is rectangular and both vertical edges of \( Q \) or both horizontal edges of \( Q \) are contained in edges of \( P \). The \textit{width} of the pipe is the distance between this pair of edges. The \textit{length} of the pipe is the distance between the other pair of edges. We say that a pipe is \textit{long} if its length is at least 3 times its width.

\textbf{Definition 2.} We say that a polyomino \( P \) has \textit{consistent parity} if all first coordinates of the corners of \( P \) have the same parity and likewise for the second coordinates. Equivalently, \( P \) has consistent parity if there exists an open \( 2 \times 2 \) square, \( S \), such that for all choices of integers \( i, j \) and \( S' = S + (2i, 2j) \), either \( S' \subseteq P \) or \( S' \cap P = \emptyset \).

Variations of the following lemma are well-known. We present a proof for completeness.

\textbf{Lemma 4.1.} Let \( P \) be an orthogonal polygon with \( n \) corners and \( h \) holes. Then, \( P \) can be divided into at most \( n/2 + h - 1 \) rectangular pieces by adding only vertical line segments to the interior of \( P \). If \( P \) is a polyomino, the rectangular pieces can be chosen to be polyominos too.
Fig. 8. A partition of a polyomino with two holes into rectangles using vertical line segments (blue).

**Proof.** For each concave corner of the polygon we add a vertical line segment in the interior of the polygon starting from that corner and going upwards or downwards (depending on the rotation of the given corner). This is illustrated in Figure 8. Let $s$ be the number of line segments added. It is easy to check that this gives a partition of $P$ into exactly $s - h + 1$ rectangles. With $h$ holes, the number of concave corners is $n/2 + 2(h - 1)$, so also $s \leq n/2 + 2(h - 1)$ and the result follows. □

Note that for a polygon with $n$ corners, $h \leq (n - 4)/4$, so we have the following trivial corollary.

**Corollary 4.2.** The number of rectangular pieces in Lemma 4.1 is at most $3/4n - 2$.

We next show that the property of consistent parity is preserved under integral offsets.

**Lemma 4.3.** Let $P$ be a polyomino. If $P$ has consistent parity, then $B(P, 1)$ and $B(P, -1)$ have consistent parity.

**Proof.** Suppose $P$ has consistent parity. Let $S$ be a $2 \times 2$ square as in Definition 2. Define $S_1 = S + (1, 1)$. It is easy to check that for all choices of integers $i, j$ and $S'_1 := S_1 + (2i, 2j)$, either $S'_1 \subseteq B(P, 1)$ or $S'_1 \cap B(P, 1) = \emptyset$. Thus, $B(P, 1)$ has consistent parity. The argument that $B(P, -1)$ has consistent parity is similar. □

**Lemma 4.4.** Let $P$ be a connected polyomino of consistent parity and without holes. Define $L_1 = B(P, 1) \setminus P$ and $L_{-1} = P \setminus B(P, -1)$. Then $G(L_1)$ and $G(L_{-1})$ both have a Hamiltonian cycle of even length.

**Proof.** To obtain a Hamiltonian cycle of $G(L_1)$, we can simply trace $P$ around the outside of its boundary, visiting all cells of $L_1$ in a cyclic order. The corresponding closed trail of $G(L_1)$ visits each vertex at least once. The assumption of consistent parity is easily seen to imply that we in fact visit each vertex exactly once, so the obtained trail is a Hamiltonian cycle. The graph $G(L_1)$ is bipartite, so the cycle has even length. The argument that $G(L_{-1})$ has a Hamiltonian cycle of even length is similar. □

With the above in hand, we are ready to state and prove our main structural results on domino packings. They are presented in Lemmas 4.5 and 4.6.

**Lemma 4.5.** Let $P$ and $P_0$ be polyominos such that $P_0 \subseteq P$, $P_0$ has no holes, and $P_0$ has consistent parity. Let the total number of corners of $P$ and $P_0$ be $n$. Define $r = \lfloor \frac{3}{8}n \rfloor$ and $Q = B(P_0, -r)$. There exists a maximum packing of $P$ with $1 \times 2$ dominos which restricts to a tiling of $Q$.

Let us briefly pause to explain the importance of Lemma 4.5. Suppose that $P$ contains a region $Q$ as described. Then Lemma 4.5 tells us that any domino tiling of $Q$ can be extended to a maximum domino packing of $P$. We can thus disregard $Q$ and focus on finding a maximum packing of $P \setminus Q$, thus reducing the problem to a smaller instance. This is one of our key tools for reducing the size of the original polyomino $P$ to a matching problem of polynomial size. The idea behind the proof...
of Lemma 4.5 is to obtain \( Q \) by peeling off a sequence of layers \( A_1, \ldots, A_r \) of \( P_0 \) (the connected components of which are Hamiltonian by the previous lemmas). We tile \( Q \) arbitrarily, tile the layers cyclically, and pack dominos in \( P \setminus P_0 \) leaving only \( O(n) \) uncovered cells. Now, we extend to a maximum packing using augmenting paths sequentially. The main observation is that the \( i \)'th augmenting path can be made to avoid \( A_{i+1}, A_{i+2}, \ldots, A_r \), and \( Q \) entirely by instead going around the cycles of \( A_i \). Moreover, we only need to do \( O(n) \) augmentations before the packing is maximum.

Here we importantly use that \( P_0 \) has no holes. If \( P_0 \) had holes, then such an augmenting path could start outside such a hole and within it, in which case it can no longer be modified to only use the outermost layer.

**Proof.** It follows from Lemma 4.3 that \( Q \) has consistent parity, and it can thus be tiled with \( 2 \times 2 \) squares and hence with dominos. Let \( Q \) be a tiling of \( Q \).

Define \( R = P \setminus P_0 \) and note that \( R \) has at most \( n \) corners. It follows from Corollary 4.2 that \( R \) can be partitioned into less than \( \frac{3}{2} n \) rectangular polyominos. Each of these rectangles has a domino packing with at most one uncovered cell (which happens when the total number of cells in the rectangle is odd). Fix such a packing \( R \) of the rectangles of \( R \) with dominos.

We next describe a tiling of \( P_0 \setminus Q \) as follows. For integers \( 1 \leq i \leq r \) we define, \( A_i = B(P_0, -i + 1) \setminus B(P_0, -i) \). Intuitively, we can construct \( Q \) from \( P_0 \) by peeling off the 'layers' \( A_i \) of \( P_0 \) one at a time. Let \( i \in \{1, \ldots, r\} \) be fixed. As \( P_0 \) has consistent parity, it follows from Lemma 4.3 that \( B(P_0, -i + 1) \) has consistent parity. It is also easy to check that \( B(P_0, -i + 1) \) has no holes either, and it then follows from Lemma 4.4 that each connected component of \( G(A_i) \) has a Hamiltonian cycle of even length. These cycles give rise to a natural tiling of \( A_i \); if \( (v_1, \ldots, v_{2k}) \) is the sequence of cells corresponding to such a cycle, then \( \{v_1 \cup v_2, v_3 \cup v_4, \ldots, v_{2k-1} \cup v_{2k}\} \) is a tiling of the cells of the cycle, and the union of such tilings over all connected components in \( G(A_i) \) gives a tiling of \( A_i \) with dominos. Denote this tiling by \( A_i \). See Figure 9 for an illustration of this construction.

Combining the tilings \( A_1, \ldots, A_r \) and \( Q \) with the packing \( R \), we obtain a domino packing \( \mathcal{P} \) of \( P \) where at most \( \frac{3}{2} n \) cells of \( P \) are uncovered. We now wish to extend this packing to a maximum packing in a way where we do not alter the tiling \( Q \) of \( Q \). If we can do this, the result will follow. Let \( M \) be the matching corresponding to \( \mathcal{P} \) in \( G(P) \). We make the following claim.

**Claim.** Let \( k \leq r \). Suppose that the matching \( M \) can be extended to a matching of size \( |M| + k \). Then this extension can be made using a sequence \( C_1, \ldots, C_k \) of \( k \) augmenting paths one after the other (that is, \( C_i \) is an augmenting path after the matching has been extended using \( C_1, \ldots, C_{i-1} \)) such that for each \( i \in \{1, \ldots, k\} \), we have that \( C_i \) only uses vertices of \( G(R \cup \bigcup_{j=1}^{i-1} A_j) \).
Before proving this claim, we first argue how the result follows. Since there are less than \( \frac{3}{4} n \) unmatched vertices in \( M \), we can extend \( M \) to a maximum matching using at most \( r = \lfloor \frac{3}{8} n \rfloor \) augmenting paths. By the claim, these paths can be chosen so that they avoid the vertices of \( G(Q) \). In particular, we never alter the matching of \( G(Q) \), so the final maximum matching restricted to \( G(Q) \) is just the tiling \( Q \).

We proceed to prove the claim by induction on \( k \). The statement is trivial for \( k = 0 \), so let \( 1 \leq k \leq r \) satisfy the assumptions of the claim and suppose inductively that \( C_i, \ldots, C_{k-1} \) can be chosen such that for each \( i \in \{1, \ldots, k-1\} \), we have that \( C_i \) only uses vertices of \( G(R \cup \bigcup_{j=1}^{k-1} A_j) \). After augmenting the matching using \( C_1, \ldots, C_{k-1} \), we have only modified the matching restricted to \( G(R \cup \bigcup_{j=1}^{k} A_j) \). By Lemma 4.6, we can find an augmenting path \( C'_k \) connecting two unmatched vertices \( u, v \) of \( G(P) \). We will modify \( C'_k \) to a path \( C_k \) with \( C_k \subseteq R \cup \bigcup_{j=1}^{k} A_j \). Write \( C'_k : u = u_1, u_2, \ldots, u_{2f} = v \). Let \( D \) be a Hamiltonian cycle of one of the connected components of \( G(A_k) \); see Figure 10. If the path \( C'_k \) ever enters the vertices of \( D \), we let \( i \) be minimal such that \( u_i \in D \) and \( j \) be maximal such that \( u_j \in D \). We can now replace the subpath \( u_i, u_{i+1}, \ldots, u_j \) of \( C_k \) with part of the Hamiltonian cycle \( D \). Whether we go clockwise or counterclockwise along \( D \) depends on whether \( u_i \) is matched with \( u_{i+1} \) in a clockwise or counterclockwise fashion in \( D \). We do the same modification for every Hamiltonian cycle \( D \) corresponding to a connected component of \( G(A_k) \) that \( C'_k \) intersects. Note that each cycle \( D \) partitions the vertices \( G(P) \setminus D \) into an interior and an exterior part. Since \( P \) has no holes and \( u, v \in R \), the original path \( C'_k \) enters \( D \) from the exterior at \( u_i \) and likewise leaves \( D \) into the exterior at \( u_j \). Also note that \( Q \) is contained in the interior parts of the cycles of \( G(A_k) \). It then follows that the final resulting path \( C_k \) avoids \( Q \) and \( A_j \) for \( j > k \), so it is contained in \( R \cup \bigcup_{j=1}^{k} A_j \).

At a high level, Lemma 4.5 allows us to ‘ignore’ parts of the polyomino \( P \) with distance \( \Omega(n) \) to the boundary. In fact, this is only true if \( P \) has no holes, but we will shortly see how to ensure that a similar property holds when \( P \) has holes. In order to argue that the answer output by simple-packer is correct, we also need to argue that we can ignore long pipes (see Definition 1). This is what motivates the following lemma which intuitively yields a reduction for shortening long pipes.

**Lemma 4.6.** Let \( k, \ell \in \mathbb{N} \) with \( \ell \) even. Let \( L \subseteq [-1, 0] \times [0, k] \), \( R \subseteq [\ell, \ell + 1] \times [0, k] \) be polyominos and define \( P = L \cup R \cup ([0, \ell] \times [0, k]) \). Color the cells of the plane in a chessboard like fashion and let \( b \) and \( w \) be respectively the number of black and white cells contained in \( P \). Assume without loss of generality that \( b \geq w \). If \( \ell \geq 2k \), then the number of uncovered cells in a maximum domino packing of \( P \) is exactly \( b - w \). Moreover, there exists a maximum domino packing such that the rectangle \([k + 1, \ell - k - 1] \times [0, k]\) is completely covered and all dominos intersecting the rectangle are horizontal.
Fig. 11. (a) The polyomino $P$. (b) Shifting a notch. (c) Cancelling two notches. (d) The partial packing obtained after shifting notches downwards (one notch at a time going from bottom to top) and cancelling notches when possible (gray) and the horizontal dominos completing the packing (green).

**Remark.** We note that the condition $\ell \geq 2k$ is necessary (except for the factor of 2). Indeed, suppose that $k$ is divisible by 4 and that $L$ consists of the white cells in $[-1, 0] \times [0, k]$ below the line $l$ with $y$-coordinate $y(l) = k/2$ and $R$ consists of the black cells in $[\ell, \ell + 1] \times [0, k]$ above $l$. Then the number of black and white cells are the same, but a simple application of Hall’s marriage theorem (to the set of black cells above $l$) shows that no perfect matching exists unless $\ell \geq k/2$.

**Proof.** As each domino covers one black and one white cell, any packing will leave at least $b - w$ cells uncovered. We thus need to demonstrate the existence of a packing with exactly $b - w$ uncovered cells. To see that such a packing exists, it is very illustrative to consider Figure 11. An example of a polyomino, $P$, is illustrated in Figure 11(a). We first tile as many cells of $L$ and $R$ as possible, such that no two uncovered cells of $L$ and $R$ share an edge. We will call these uncovered cells *notches*. We next show how we can alter the configuration of notches by only adding a layer of width 2. First, we note that a notch can be shifted an even number of cells downwards or upwards using the construction in Figure 11(b). In case we have two notches of different colours in the chessboard coloring and with no other notches between them, we can use the construction in Figure 11(c) to cancel these two notches from the configuration of notches. Our goal is to use the constructions of (b) and (c) to shift the notches of $L$ and $R$ downwards, cancelling notches if possible. Going through the notches of $L$ from bottom to top, we shift them down as far as possible using the construction in (b). In case a notch has a different color than the nearest notch below it, we use construction (c) to cancel them. We further add horizontal dominos such that the configuration of notches is preserved at all other positions than where the shifting or cancelling occurs. We proceed similarly for $R$. The process from start to end is illustrated in Figure 11(d) which also shows the resulting partial tiling. The red lines in the figure separate the steps of the process.
Fig. 12. A contraction of the algorithm simple-packer with one fat and two skinny rectangles. The algorithm moves all corners \( c_{i+1}, \ldots, c_n \) to the left, essentially contracting the area between the vertical lines \( v'_1 \) and \( v'_2 \) to nothing.

Note that each added layer of the process has thickness 2. Initially, each of \( L \) and \( R \) consists of at most \( \lceil k/2 \rceil \) notches (after the first step which isolates the notches). Moreover, if \( k \) is odd and there is \( \lceil k/2 \rceil \) notches in \( L \) or \( R \), then no shifting/cancelling is needed on that side. It then follows from the assumption \( l \geq 2k \) that we are able to finish this partial packing. Let \( b' \) and \( w' \) be the number of black and white cells uncovered by this partial packing. Then \( b' - w' = b - w \). It is also easy to check that we can complete the packing of \( P \) using only horizontal dominos and leaving exactly \( b' - w' = b - w \) cells uncovered. This completes the proof.

\[ \square \]

4.4 Correctness of the Algorithm

We now verify that the number of uncovered cells in maximum packings is invariant under a single contraction, and the correctness of the algorithm hence follows. To this end, suppose that \( x(c_{i+1}) - x(c_i) > 9n \), so that we move the corners \( c_{i+1}, \ldots, c_n \) to the left; see Figure 12. It is clear that a domino packing with exactly \( \ell \) uncovered cells after the contraction gives rise to a domino packing with exactly \( \ell \) uncovered cells before the contraction, simply by inserting extra horizontal dominos in the rectangles that were contracted. For the converse, let \( v_1 \) and \( v_2 \) be vertical lines with \( x \)-coordinates \( x(c_i) \) and \( x(c_{i+1}) \), respectively, and let \( V \) be the vertical strip bounded by \( v_1 \) and \( v_2 \). The intersection \( P \cap V \) is a collection of disjoint rectangles \( R_1, \ldots, R_k \) of width \( x(c_{i+1}) - x(c_i) \) and various heights. We define a rectangle \( R_i \) to be fat if its height is more than \( 3n \), and otherwise \( R_i \) is skinny. We now define a polyomino \( P_0 \) in order to apply Lemma 4.5. For each fat rectangle \( R_i \), we let \( R'_i \subseteq R_i \) be the maximum rectangle with even coordinates and add \( R'_i \) to \( P_0 \). As each rectangle \( R_i \) corresponds to exactly two horizontal edges, the number \( k \) of rectangles is at most \( n/4 \) and in particular, the number of corners of \( P \setminus P_0 \) is at most \( 2n \). Letting \( Q := B(P_0, -\lfloor 3n/2 \rfloor) \), we get from Lemma 4.5 that there exists a maximum packing of \( P \) that restricted to \( Q \) is a tiling.
We define \( P_1 := P \setminus Q \) and observe that the contraction corresponds to contracting a set of long pipes in \( P_1 \). These pipes are the skinny rectangles \( R_i \) and the parts of the fat rectangles vertically above and below the removed part \( Q \). We therefore get from Lemma 4.6 that a maximum packing before the contraction having \( \ell \) uncovered cells, gives rise to a packing of the contracted polyomino with exactly \( \ell \) uncovered cells.

5 FASTER DOMINO PACKING ALGORITHM

We will next describe the steps of our faster, albeit more complicated, algorithm, for finding the maximum domino packing of a polyomino \( P \). Figures 13–15 demonstrate the steps on a concrete polyomino \( P \).

Step 1: Compute the unique maximal polyomino \( P_1 \subseteq P \) with all coordinates even. We define \( P_1 \) to be the union of all \( 2 \times 2 \) squares \( S \) of the form \( S = [2i, 2i + 2] \times [2j, 2j + 2] \) with \( i, j \in \mathbb{Z} \) and \( S \subseteq P \). See the upper left and bottom part of Figure 13. It is readily checked each corner of \( P_1 \) corresponds to a unique corner of \( P \), and thus \( P_1 \) has at most \( n \) corners. As we will see, \( P_1 \) can be computed in time \( O(n \log n) \).

Step 2: Compute a polyomino \( P_2 \subseteq P_1 \) with no holes and consistent parity (Definition 2) by carving channels in \( P_1 \). Define \( P'_0 := P_1 \). For \( i = 0, 1, \ldots \), we do the following. If there are holes in \( P'_i \), we find a set of minimum size of \( 2 \times 2 \) squares \( S_1, \ldots, S_k \) contained in \( P'_i \) and with even corner coordinates.
that connects an edge of a hole to an edge of the outer boundary of \( P'_i \). To be precise, an edge of \( S_1 \) should be contained in the boundary of a hole of \( P'_i \), an edge of \( S_k \) should be contained in the outer boundary of \( P'_i \), and for each \( j \in \{1, \ldots, k-1\} \), \( S_j \) and \( S_{j+1} \) should share an edge. We choose these squares so that they together form an L-shape (which may degenerate to just a \( 2 \times 2 \)-or a \( 2k \times 2 \)-rectangle), which is always possible: An arbitrary shortest path of such squares can be replaced by an L-shaped path of the same length (keeping the first and last squares unchanged), and if this is not contained in \( P'_i \), there would exist an even shorter one, which is a contradiction.

We then define the polyomino \( P'_{i+1} := P'_i \setminus \bigcup_{j=1}^k S_j \), which has fewer holes than \( P'_i \). We stop when there are no more holes and define \( P_2 := P'_1 \) to be the resulting hole-free polyomino. Note that in iterations \( i \geq 1 \), the holes may get connected to holes that were eliminated in earlier iterations or to channels carved in earlier iterations. See the upper right part of Figure 13. We will later see that \( P_2 \) has strictly fewer \( 0 \) than \( 3n \) corners and that it can be computed in time \( O(n^3) \).

**Step 3:** Compute the offset \( Q := B(P_2, -[3n/2]) \) and then \( P_3 := P \setminus Q \). See the left part of Figure 14. Note that we remove \( Q \) from the original polyomino \( P \) in order to get \( P_3 \), and not from \( P_2 \). It is easy to check that \( Q \) has at most \( 3n \) corners and consistent parity by Lemma 4.3. Hence, \( P_3 := P \setminus Q \) has at most \( 4n \) corners and, as we will see, \( P_3 \) has the property that for any \( x \in P_3 \), we have \( \text{dist}(x, \partial P_3) = O(n) \). We will show how this step can be carried out in time \( O(n \log n) \).

**Step 4:** Find the long pipes of \( P_3 \). Find all maximal long pipes \( T_1, \ldots, T_r \) in \( P_3 \) (recall that a pipe is long if its length is at least \( 3 \) times its width). See the right part of Figure 14. As we will see, there are at most \( O(n) \) such pipes, their interiors are disjoint, and they each have width \( O(n) \). Later we will show how the pipes can be found in time \( O(n \log n) \).

**Step 5:** Shorten the pipes and compute the associated graph \( G^* \). Define \( G_3 := G(P_3) \). We modify \( G_3 \) by performing the following shortening step for each \( 1 \leq i \leq r \); see Figure 15. Assume with no loss of generality that the pipe \( T_i \) is of the form \( T_i = [0, \ell] \times [0, k] \) where \( \ell \) is the length and \( k \leq \ell/3 \) is the width. If \( \ell \leq 6 \), we do nothing. Otherwise, for each \( j \in \{0, \ldots, k-1\} \), we let \( S_j = [k+2, r] \times [j, j+1] \), where \( r := 2[\ell/2] - k - 2 \), so that \( G(S_j) \) is a horizontal path in \( G_3 \) consisting of an even number of vertices. For each \( j \in \{1, \ldots, k-1\} \), we proceed by deleting the vertices of \( S_j \) and their incident edges from \( G_3 \), and instead, we add an edge from the cell \( [k+1, k+2] \times [j, j+1] \) to the cell \( [r, r+1] \times [j, j+1] \) (i.e., we connect the cells to the left and right of \( S_j \) with each other).
We denote the graph obtained after iterating over all \( i \) by \( G^* \). Note that in \( G^* \), there are only \( O(k^2) = O(n^2) \) vertices corresponding to cells in each pipe \( T_i \), since each pipe has width \( k = O(n) \). We show below that \( G^* \) has \( O(n^3) \) vertices and can be computed in time \( O(n^3) \).

**Step 6: Find the size of a maximum domino packing of \( P \).** We finally run a maximum matching algorithm on \( G^* \). The algorithm outputs \( |M| + \frac{\text{area}(P) - |V(G^*)|}{2} \), where \( M \) is a maximum matching in \( G^* \) and \( V(G^*) \) are the vertices of \( G^* \).

This completes the description of the algorithm. In Section 5.1, we will use the results of Section 4.3 to argue that the algorithm works correctly. In Section 5.2, we will show that the reduced graph \( G^* \) has \( O(n^3) \) vertices and edges. Finally, in Section 5.3, we will use this to argue how the steps of the algorithm can be implemented with the claimed running times.

We need to argue that the algorithm outputs the correct value and that the different steps can be implemented to obtain the stated running times.

### 5.1 Correctness of the Fast Algorithm

We now show that the algorithm correctly finds the size of a maximum domino packing. To show this, it suffices to show that maximum matchings of \( G_0 = G(P) \) and \( G^* \) leave the same number of unmatched vertices. First note that \( P_1 \) has at most \( n \) corners. Further, a polyomino with \( n \) corners can have at most \((n - 4)/4 \) holes, and since we remove a hole and add at most 6 new corners in going from \( P_i \) to \( P_{i+1}' \), \( P_2 \) has at most \( 5n/2 < 3n \) corners. It follows that also \( Q = B(P_2, \lfloor 3n/2 \rfloor) \) has at most \( 3n \) corners. Letting \( n_1 \) denote the number of corners of \( P_3 = P \setminus Q \) it finally follows that \( n_1 \leq 4n \). Now \( \text{dist}(\partial P, Q) \geq \lfloor 3n/2 \rfloor \geq \lfloor 3n/1 \rfloor = \lfloor 3n/8 \rfloor \) and moreover, \( Q \) has consistent parity and no holes, so Lemma 4.5 applies, giving that \( G_0 \) has a maximum matching, which restricts to a perfect matching of \( G(Q) \) and to a maximum matching of \( G_3 \). In particular, maximum matchings of \( G_0 \) and \( G_1 \) leave the same number of vertices unmatched.

Next, we argue that maximum matchings of \( G_3 \) and \( G^* \) again leave the same number of vertices unmatched. It is easy to see that a maximum matching of \( G^* \) can be extended to a matching of \( G_3 \) with the same number of unmatched vertices by simply inserting more horizontal dominos in the horizontal pipes and vertical dominos in the vertical pipes (here we use that the \( S_j \)'s as defined in step 3, each consists of an even number of cells). Conversely, let \( M_1 \) be a maximum matching of \( G_3 \). We show that \( G^* \) has a matching \( M_2 \) with the same number of uncovered cells. For this we consider the pipes \( (T_i)_i \) found in step 4 of the algorithm. Note first that their interiors are disjoint. To see this, note that the pipes are of two types, vertical or horizontal depending on whether their vertical or horizontal boundary edges are contained in \( P_3 \). Now it is simple to check that two pipes of different types must be interior disjoint. The fact that two pipes of the same type are interior disjoint follows from the pipes being chosen to be maximal. For each \( 1 \leq i \leq r \), we let \( T_i' \) be the pipe obtained from \( T_i \) by shortening \( T_i \) by one layer of cells in each end. The length of \( T_i' \) is thus shorter by two than that of \( T_i \). Let further \( L_i \supseteq T_i' \) consist of all cells of \( P \) which are covered by a domino which covers at least one cell of \( T_i' \). The interiors of the sets \( (L_i)_i \) are

pairwise disjoint and each $L_i$ is of the form of the set $L$ in Lemma 4.6 (up to a 90 degree rotation). Moreover, the maximum matching $M_1$ restricts to a maximum matching of $M'_1$ of $G(P_3 \setminus \bigcup_{i=1}^r L_i)$ and a maximum matching $M^{(i)}_1$ of $G(L_i)$ for $1 \leq i \leq r$. For $1 \leq i \leq r$, we let $G^{(i)} = G(L_i)$ and $G^{(i)}_2$ be the corresponding subgraph of $G_2$. We define $M_2$ to be $M'_1$ combined with any maximum matchings of the $G^{(i)}_2$, $1 \leq i \leq r$. By applying Lemma 4.6 to each $L_i$, it follows that the maximum matchings of $G^{(i)}_3$ and $G^{(i)}_2$ leave the same number of unmatched vertices. It thus follows that $M_2$ and $M_1$ leave the same number of unmatched vertices. This finishes the argument that the algorithm works correctly.

5.2 Bounding the Size of the Reduced Instance

In determining the running time of our algorithm, it is crucial to bound the size of the reduced instance $G^*$. In this section, we show that $G^*$ has $O(n^3)$ vertices. As explained in the next section, we can then find a maximum matching of $G^*$ in $O(n^3 \text{polylog } n)$ time.

We start out by proving the following lemma.

**Lemma 5.1.** The polyomino $P_3$ contains no $63n \times 63n$ square as a subpolyomino.

**Proof.** Let $n' = \lfloor 3n/2 \rfloor$. We show that $P_3$ contains no $41n' \times 41n'$ square as a subpolyomino and the desired result will follow. Suppose for contradiction that $S \subseteq P_3$ is such a subpolyomino. Note that $Q$ consists of exactly those points of $P_1$ of distance at least $n'$ to all the channels of $C := P_1 \setminus P_2$ and to $\partial P_1$. Thus, any point $x \in P_3$ has distance at most $n'$ to $C$ or to $\partial P_1$. In particular, $S$ contains a $39n' \times 39n'$ square subpolyomino $S_1 \subseteq P_1$ all points of which are of distance at least $n'$ to $\partial P_1$, and thus of distance at most $n'$ to $C$.

By the way we chose the channels, each channel connects a hole of $P_1$ with either the boundary of $P_1$ or with a channel already carved in an earlier iteration. Since $\partial P_1 \cap S_1 = \emptyset$, it follows that any channel intersecting $S_1$ has an end outside $S_1$ and thus leaves $S_1$ through an edge of $S_1$.

Write $S'_1$ for the central $3n' \times 3n'$ square polyomino of $S_1$; see Figure 16(a). Since any point of $S_1$ is of distance at most $n'$ to $C$, $S'_1$ must intersect a channel $C_1 \subseteq C$ (depicted in red in the figure). We know that $C_1$ leaves $S_1$. It is simple to check that this leads to the existence of a $19n' \times 18n'$ rectangular polyomino $S_2 \subseteq S_1$ having along one of its sides a straight part of the channel $C_1$ of length $18n'$; see Figure 16(a). Assume with no loss of generality that $S_2 = [0, 18n'] \times [0, 19n']$ and that the channel $C_1$ runs along the base of the rectangle $S_2$ as in 16(a). For $1 \leq i \leq 6$ we define $S^{(i)}_2$.

![Figure 16. Situations in the proof of Lemma 5.1.](image-url)
to be the square polyomino \([3(i - 1)n', 3n'] \times [0, 3n']\). By the same reasoning as above, each of these squares must intersect the set of channels \(C\) non-trivially. As each channel turns at most once by construction, the squares \(S_2^{(i)}\) are disjoint from \(C_1\). To finish the proof, we require the following claim.

**Claim.** Let \(B = [0, k] \times [0, \ell], k, \ell \in \mathbb{N}\) be a \(k \times \ell\) square polyomino. Suppose that \([0, k] \times [-1, 0]\) is contained in some channel \(C'\), and that \([0, k] \times [\ell, \ell + 1]\) is contained in some other channel \(C''\). Then \(k \leq \ell + 2\).

**Proof of Claim.** See Figure 16(c). Suppose without loss of generality that \(C'\) was carved in iteration \(i\) and \(C''\) was carved in iteration \(j\) in the process of generating \(P_2\) in step 2 of the algorithm, and that \(i < j\). The channel \(C''\) was chosen to be the union of a minimum number of \(2 \times 2\) squares with even corner coordinates such that these squares formed a ‘path’ connecting an edge of yet unconnected hole \(H\) of \(P_{j-1}'\) with an edge of the outer boundary of \(P_{j-1}'\) (see step 2 of the algorithm for a precise definition). At the time \(C''\) was carved, \(C'\) had already been carved and thus the edges of \(C'\) (except the two "ends" of length 2) are part of the outer face of \(P_{j-1}'\). Under the assumption \(k > \ell + 2\), we can find a path connecting \(H\) to \(\partial P_{j-1}'\), using fewer \(2 \times 2\) squares; see the bottom part of Figure 16(c). In turn, this implies that we could have picked a channel \(\tilde{C}''\) in place of \(C''\) consisting of fewer \(2 \times 2\) squares. This is a contradiction, so we conclude that \(k \leq \ell + 2\). \(\square\)

Let us now finish the proof of the lemma. We know that the two squares \(S_2^{(2)}\) and \(S_2^{(5)}\) intersect channels of \(C\). Let us denote these not necessarily distinct channels respectively \(C_2\) and \(C_3\). If these channels are the same, the channel \(C_2\) passes straight through \(S_2^{(5)}\). But then the two channels \(C_1\) and \(C_2\) run in parallel for a length of at least \(3n' + 4\) and they have distance at most \(3n'\) which gives a contradiction with the claim. Thus, \(C_2\) and \(C_3\) are different channels. By the same reasoning as for \(C_1\), the channel \(C_2\) must leave \(S_2\). If it does so in a direction parallel to \(C_1\), we similarly obtain a contradiction with the claim. Thus, it must leave \(S_2\) in a direction perpendicular to \(C_1\). The same logic applies to \(C_3\); see Figure 16(b). Now the two channels \(C_2\) and \(C_3\) provide a contradiction to the claim. Indeed, their straight segments span a box \(S_3\) of dimensions \(\ell \times 18n'\) where \(\ell \leq 18n' - 4\). With this contradiction, we conclude that \(P_3\) contains no \(41n' \times 41n'\) square as a subpolyomino and the proof is complete. \(\square\)

**Remark.** No serious effort has been made to optimize the constants in Lemma 5.1.

**Corollary 5.2.** For any \(x \in P_3\), we have \(\text{dist}(x, \partial P_3) \leq 32n\).

**Proof.** If not, \(P_3\) contains a \(63n \times 63n\) square, a contradiction. \(\square\)

Corollary 5.2 shows that each point of \(P_3\) is of distance \(O(n)\) to the boundary of \(P_3\). In particular, this shows that the long pipes \(T_1, \ldots, T_r\) found in step 4 of the algorithm all have width at most \(O(n)\). The following lemmas allow us to give a bound on \(r\), i.e., the number of pipes.

**Lemma 5.3.** Let \(G = (V, E)\) be a graph of order \(n \geq 2\) with no self-loops but potential multiple edges. Suppose that \(G\) has a planar embedding such that for any pair of multiple edges \((e_1, e_2)\), the Jordan curve formed by \(e_1\) and \(e_2\) in the planar embedding of \(G\) contains a vertex of \(G\) in its interior. Then the number of edges of \(G\) is upper bounded by \(3n - 5\).

**Proof.** In what follows, we will use the classic result that the number of edges of a simple planar graph of order \(n \geq 3\) is upper bounded by \(3n - 6\). We prove the result by strong induction on \(n\). For \(n = 2\) the result is trivial so let \(n > 2\) be given and suppose the bound holds for smaller values of \(n\). Let \(\mathcal{E}\) be a planar embedding of \(G\). Let \((e_1, e_2)\) be a pair of distinct multiple edges that is minimal in the sense that no other such pair \((e_1', e_2')\) exists with the following property: If \(y\) and \(y'\)
are the Jordan curves formed by \((e_1, e_2)\) and \((e'_1, e'_2)\) in \(E\), then \(\text{Int } \gamma' \subseteq \text{Int } \gamma\). Assume that \(e_1\) and \(e_2\) connect vertices \(u\) and \(v\). Let \(V'\) be the set of vertices of \(G\) that are contained in \(\text{Int } \gamma\) under \(E\) and let \(k = |V'|\). Then, \(1 \leq k \leq n - 2\). Let \(G_1 = (V_1, E_1)\) where \(V_1 = V' \cup \{u, v\}\) and \(E_1\) is formed by \(e_1\) together with all edges of \(G\) that are incident to a vertex in \(V'\). Let \(G_2 = (V_2, E_2)\) where \(V_2 = V \setminus V'\) and \(E_2 = E \setminus E_1\). Clearly \(G_1\) is a simple planar graph on \(k + 2\) vertices. Moreover, it is readily checked that \(G_2\) is a planar graph on \(n - k\) vertices which satisfies the assumptions of the lemma. Note that \(2 \leq n - k < n\). It thus follows from the inductive hypothesis that the number of edges of \(G\) is upper bounded by

\[
3(n - k) - 5 + 3(k + 2) - 6 = 3n - 5.
\]

This completes the proof. □

**Lemma 5.4.** Let \(P\) be a polyomino with \(n\) corners. Let \(Q_1, \ldots, Q_3\) be pairwise disjoint pipes of \(P\), no two of which are contained in a larger pipe of \(P\). In other words, for distinct \(i, j\), there exists no pipe \(Q\) with \(Q_i \cap Q_j \subseteq Q\). Then \(s \leq 3n - 5\).

**Proof.** We construct a graph \(G = (V, E)\) as follows. We define \(V\) to be the set of (geometric) edges of \(P\). For each \(i \in \{1, \ldots, s\}\) we let \(u_i, v_i \in V\) be the two parallel edges of \(P\) which contain two opposite sides of \(Q_i\) and we add the edge \((u_i, v_i)\) to \(E\). Thus, \(G\) is a graph of order \(n\) with exactly \(s\) edges. We note that \(G\) may have multiple edges but it has a natural planar embedding \(E\) such that for each pair of multiple edges \((e_1, e_2)\) the Jordan curve formed by \(e_1\) and \(e_2\) under \(E\) contains a vertex of \(G\) in its interior (see Figure 17). Here we used that for any two \(i, j\) with \(1 \leq i < j \leq s\), the two pipes \(Q_i\) and \(Q_j\) are not contained in a larger pipe of \(P\). It thus follows from Lemma 5.3 that \(s \leq 3n - 5\). □

By Lemma 5.4, the number of pipes is \(r = O(n)\), so when performing the shortening reduction in step 5 of our algorithm, the part of \(G^*\) contained in contracted pipes gets size \(O(n^3)\). We finish this section by showing that \(P_3 \setminus \bigcup_{i=1}^{j'} T_j\) also consists of \(O(n^3)\) cells. From this it will follow that \(G^*\) is of order \(O(n^3)\), which is what we require. We state the result as a lemma.

**Lemma 5.5.** The reduced instance \(G^*\) found by our algorithm has \(O(n^3)\) vertices and edges.

**Proof.** For technical reasons to be made clear shortly we define \(T'_j\) to be the pipe obtained from \(T_j\) by shortening \(T_j\) by a layer of cells in each end. The length of \(T'_j\) is thus exactly the length of \(T_j\) minus 2. Let \(R\) be the polyomino \(P_3 \setminus \bigcup_{i=1}^{j'} T'_i\). Consider a (geometric) edge \(e\) in the set \(\partial R \setminus \partial P_3\) which is an edge forming an end of a shortened pipe \(T'_j\). It then follows that the endpoints of \(e\) are corners of \(R\) and in particular that \(e\) is not contained in a longer edge of \(\partial R\) (this would not necessarily be the case if we had not shortened the pipes a layer in each end when defining \(R\)). We will use this observation shortly.

As discussed, it suffices to show that \(R\) has \(O(n^3)\) cells. Note that \(R\) has \(O(n)\) corners: Indeed, \(R\) is obtained from \(P_3\), which has \(O(n)\) corners, by removing \(O(n)\) pipes, each of which adds only
4 corners. We show that any point $x \in R$ is of distance $O(n)$ from a corner. Since each corner can have at most $O(n^2)$ cells within distance $O(n)$, this will show that $R$ has $O(n^3)$ cells.

So let $x \in R$ be arbitrary. Also let $c := 32$. By Corollary 5.2, any point of $P_3$ is of distance at most $cn$ to $\partial P_3$. It follows that, similarly, any point of $R$ is of distance at most $cn$ to $\partial R$. Let $S$ be the $6cn \times 6cn$ square centered at $x$ and suppose that $S$ contains no corner of $R$. Then $\partial R \cap S$ is a collection of horizontal and vertical straight line segments. Moreover, they are either all horizontal or all vertical as otherwise, they would intersect in a corner of $R$ inside $S$. Assume without loss of generality that they are all horizontal. Using that any point of $R$ is of distance at most $cn$ to $\partial R$, it follows that there exists two such parallel segments of distance at most $2cn$, one being above $x$ and one being below. Take a closest pair of such segments. Together they form a pipe $T$ of $R$ of length $6cn$ and width at most $2cn$, i.e., a pipe of a length at least three times its width. Now $T$ is disjoint from the pipes $T_1, \ldots, T_r$ (since $T \subseteq R$ and each $T'_i$ is disjoint from $R$). Then, $T$ is a pipe of $R$ but we claim that it is in fact also a pipe of $P_3$. To see this, we note that by Corollary 5.2, the pipes found in step 4 of our algorithm have width at most $2cn$. In particular, the edges of $\partial R \setminus \partial P_3$ have length at most $2cn$ and we saw that they are not contained in longer edges of $\partial P_3$. However, the pipe $T$ has length $6cn$ and so, the long edges of $T$ are in fact edges of $\partial P_3$, so $T$ is a pipe of $P_3$. This contradicts the maximality of the set of pipes $T_1, \ldots, T_r$. We thus conclude that $S$ must contain a corner of $\partial R$. Since $x \in R$ was arbitrary, this shows that any point in $R$ is of distance at most $3cn = O(n)$ to a corner of $R$ and the proof is complete. 

5.3 Implementation of the Individual Steps

We next describe how the different steps of the domino packing algorithm fast-packer can be implemented.

Step 1: Compute the unique maximal polyomino $P_1 \subseteq P$ with all coordinates even. We first compute the set $P_1 \subseteq P$ with consistent parity. To obtain $P_1$, we move all corners of $P$ to the interior of $P$ to the closest points with even coordinates as shown in Figure 13.

Moving the corners may cause some corridors of $P$ to collapse (namely the corridors of $P$ of thickness 1), so that $P_1$ has overlapping edges corresponding to degenerate corridors. The degenerate vertical corridors can be filtered out in $O(n \log n)$ time as follows (and the degenerate horizontal ones are handled analogously). We sort the vertical edges after their $x$-coordinates and thus partition the edges into groups with identical $x$-coordinates. For each group of vertical edges with the same $x$-coordinate, we sort them according to the $y$-coordinates of their lower endpoints. Let $e_1, \ldots, e_k$ be one such sorted group. We run through the edges $e_1, \ldots, e_k$ in this order. For each edge $e_i$, we run through the succeeding edges until we get to an edge $e_j$ which is completely above $e_i$. For each of the overlapping edges $e_k, k \in \{i + 1, \ldots, j - 1\}$, we remove the corresponding degenerate corridor of $P_1$ created by $e_i$ and $e_k$. Since no triple of edges can be pairwise overlapping, there are $O(n)$ overlapping pairs in total, so this process is dominated by sorting, which takes $O(n \log n)$ time.

Step 2: Compute a polyomino $P_2 \subseteq P_1$ with no holes and consistent parity (Definition 2) by carving channels in $P_1$. Remember that we need to find a set of minimum size of $2 \times 2$ squares $S_1, \ldots, S_k$ contained in $P'_i$ and with even coordinates that connects an edge of a hole to an edge of the outer boundary of $P'_i$. For each pair of an edge of a hole of $P'_i$ and an edge of the outer boundary of $P'_i$, we can compute the size of the smallest set of squares connecting those two edges in $O(1)$ time, so by checking all pairs, we find the overall smallest set in $O(n^2)$ time. Note that no edge of a middle square $S_j, 2 \leq j \leq k - 1$, is contained in the boundary $\partial P'_i$, since otherwise, there would be a smaller set of squares with the desired properties. Therefore, constructing $P'_{i+1} := P'_i \setminus \bigcup_{j=1}^k S_j$ can...
then be done in $O(1)$ time once the squares $S_j$ have been found. Since there are initially $O(n)$ holes to eliminate, the process takes $O(n^3)$ time in total.

**Step 3:** Compute the offset $Q := B(P_2, −[3n/2])$ and then $P_3 := P \setminus Q$. We now explain how to compute the set $Q \subseteq P_2$ defined by $Q := B(P_2, −[3n/4])$. The boundary of $Q$ can be computed from the $L_∞$ Voronoi diagram $VD := VD(P_2)$ of the edges of $P_2$ by a well-known technique described by Held, Lukács, and Andor [18], as follows. The Voronoi diagram $VD$ is a plane graph contained in $P_2$ that partitions $P_2$ into one region $R(e_i)$ for each edge $e_i$ of $P_2$, such that if $x \in R(e_i)$ then $\text{dist}(x, e_i) = \text{dist}(x, ∂P_2)$, and $VD$ consists of horizontal and vertical line segments and line segments that make $45^\circ$ angles with the $x$-axis; see Figure 18 (left).

We find all maximal subgraphs of $VD$ consisting of points with distance at least $[3n/4]$ to $∂P_2$. The leaves of each subgraph $G$ lie on a cycle in $P_2$ where the distance to $∂P_2$ is constantly $[3n/4]$, and the cycles can be found by traversing the leaves of $G$ in clockwise order; see [18] for the details and Figure 18 (right) for a demonstration.

We can compute $VD$ in $O(n \log n)$ time using the sweep-line algorithm of Papadopoulou and Lee [30]. In our special case where all edges are horizontal or vertical, the algorithm becomes particularly simple as described by Martínez, Vigo, Pla-García, and Ayala [24]. Once we have $VD$, it takes $O(n)$ time to compute $∂Q$.

One can avoid the computation of $VD$ by offsetting the boundary into the interior by distance $1$ repeatedly $[3n/4]$ times. After each offset, we remove collapsed corridors as described in step 1. This would take in total $O(n^2 \log n)$ time.

The representation of $P_3 := P \setminus Q$ is obtained by simply adding the cycles representing the boundary of $Q$ to the representation of $P$.

**Step 4:** Find the long pipes of $P_3$. Recall that each long pipe $T_i \subseteq P_3$ is a maximal rectangle with a pair of edges contained in $∂P_3$ which are at least $3$ times as long as the other pair of edges.

To find the pipes, we compute the $L_∞$ Voronoi diagram $VD_1 := VD(P_3)$ of the edges of $P_3$ in $O(n \log n)$ time, as described in step 3. Consider a long pipe $T_i$. Assume without loss of generality that $T_i = [0, \ell] \times [0, k]$ where $\ell$ is the length and $k \leq \ell/3$ is the width. We now observe that the segment $e := [k/2, \ell − k/2] \times (k/2)$ in the horizontal symmetry axis of $T_i$ is contained in an edge of $VD_1$, since for any point $p$ in $e$, the edges of $P_3$ closest to $p$ are the horizontal edges of $T_i$.

Each horizontal or vertical edge $e$ of $VD_1$ separates the regions of points that are closest to a pair $s_1, s_2$ of horizontal or vertical edges of $P_3$. It is easy to check whether $s_1, s_2$ define a long pipe containing (a part of) $e$. Hence, all pipes can be identified by traversing the edges of $VD_1$. As $VD_1$ has complexity $O(n)$, this step takes $O(n \log n)$ time in total.
Step 5: Shorten the pipes and compute the associated graph $G^\ast$. Recall that we define $G_3 := G(P_3)$ and obtain the final graph $G^\ast$ by replacing long horizontal (resp. vertical) paths in pipes with long horizontal (resp. vertical) edges. Once $P_3$ and the long pipes have been computed, it is straightforward to construct $G^\ast$ in $O(n^3)$ time, since the size of $G^\ast$ is $O(n^3)$ by Lemma 4.1.

Step 6: Find the size of a maximum domino packing of $P$. Our algorithm outputs $|M| + \frac{\text{area}(P) - |V(G^\ast)|}{2}$, where $M$ is a maximum matching in $G^\ast$ and $V(G^\ast)$ are the vertices of $G^\ast$. It therefore remains to find a maximum matching in $G(P^\ast)$, and we can use the algorithms from the papers cited in the introduction.

Remark. We note that we can also find an (implicit) description of a maximum domino packing of $P$. We simply extend the matching of $G^\ast$ by inserting horizontal dominos in the horizontal pipes and vertical dominos in the vertical pipes. We further choose any standard tiling for $Q$, e.g., the one that uses only horizontal dominos. It follows from the correctness of the algorithm that this gives a maximum domino packing of $P$.

REFERENCES


[27] Shay Mozes and Christian Wulff-Nilsen. 2010. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log \log n)$ time. In *16th Annual European Symposium on Algorithms (ESA’10)*. https://doi.org/10.1007/978-3-642-15781-3_18


Received 19 April 2022; revised 13 March 2023; accepted 14 March 2023