The 2×2 Simple Packing Problem

André van Renssen* Bettina Speckmann†

Abstract

We significantly extend the class of polygons for which the 2×2 simple packing problem can be solved in polynomial time.

1 Introduction

We study the 2×2 simple packing problem: Given a simple rectilinear grid polygon P consisting of n edges and containing N grid cells, determine the maximum number of non-overlapping, axis-aligned 2×2 squares that can be packed into P [4]. The optimal solution is not necessarily unique with respect to the placement of the squares (Fig. 1). We, however, are interested only in the optimal number of squares.

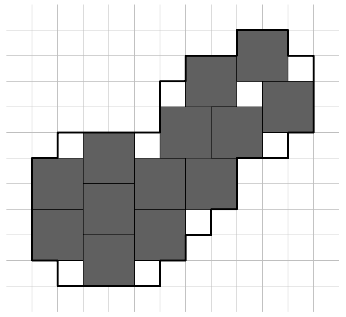

Figure 1: A polygon and one of its optimal solutions.

El-Khechen [6] proved that there always exists an optimal solution such that all packed squares are aligned on the grid. Hence we consider only optimal solutions of this type. We say that placing a square in a certain location is optimal, if there exists an optimal solution such that a square is placed in that location. Similarly, we say that not placing a square in a certain location does not affect optimality, if there is an optimal solution, such that no square is placed in that location.

Previous work. For polygons with holes, the 2×2 simple packing problem is known to be NP-hard [1]. Recently, it was proven to be NP-complete [5]. When the polygon does not contain holes it is not known whether the problem is NP-complete. An $O(N)$ time 1/2-approximation algorithm is known to exist [2]. This algorithm can be modified to run in $O(n^2)$ time by performing a line sweep from bottom to top [8].

For polygons without holes there exists a PTAS [7] that runs in $O(k^2 N^{\frac{1}{\epsilon^2}}/\epsilon^2)$ time, where k is the size of the square (in our case $k = 2$) and $\epsilon > 0$. The algorithm has an error ratio of at most $(1+\epsilon)^2$. A faster PTAS [3] runs in $O(\widehat{N} \log \widehat{N} + \widehat{N} \Delta^{\frac{1}{\epsilon}-1})$ time, where $\widehat{N}$ is the number of possible locations to place squares, Δ is the number of squares any point in the 2-dimensional plane can be in (in our case $\Delta = 4$), and $\epsilon > 0$. The algorithm has an error ratio of at most $1 + \epsilon$.

For a few classes of polygons, namely staircases, pyramids, and Manhattan skyline polygons, the problem is solvable in $O(n)$ time [6]. These classes of polygons can be solved by filling them from bottom to top with squares placed in the corners.

Results. We describe a different approach: instead of placing squares, we determine where *not* to place squares. This allows us to solve a number of classes of polygons in $O(n^2)$ time. Our polygon classes are significantly larger than those previously solvable and include staircases, pyramids, and Manhattan skyline polygons.

In Section 2 we present some observations on parts of polygons that cannot contain squares. In Section 3 we consider the intersection graph of all possible squares and we present a number of rules to reduce this graph. In Section 4 we describe a class of polygons that can be solved based on these reduction rules. In Section 5 we describe a second class of polygons which can be solved more efficiently, without considering the intersection graph. Neither of these polygons classes is contained in the other. Additional details and extensions can be found in the Master's thesis of the first author [8].

2 Tight Corridors and Tails

The dual graph of a grid polygon P has a vertex for each cell of P and an edge between two vertices iff their corresponding cells share a border. A *tight corridor* of P corresponds to a path in the dual graph, where each vertex has degree at most 2 and no vertex is part of a 4-cycle. A *tail* is a tight corridor in which at least one vertex has degree 1. We prove in [8] that no cell of a tight corridor can be part of a square. We also show

*School of Computer Science, Carleton University, Canada, avrensse@connect.carleton.ca This research was partially supported by NSERC.

†Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands, speckman@win.tue.nl Bettina Speckmann is supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.022.707.

how to remove these cells in $O(n)$ time. Every solvable class of polygons can be extended by attaching tails.

3 The Intersection Graph

Instead of placing squares, we determine where not to place squares. For this, we use the *intersection graph* $G = (V, E)$ of all possible squares (see Fig. 2). Each square is represented in G by a vertex located at the position of its center on the grid. Two vertices are connected by an edge iff their corresponding squares overlap. Clearly every vertex of G has degree at most 8. El-Khechen [6] already observed that solving the Maximum Independent Set Problem on G results in an optimal non-overlapping placement of squares.

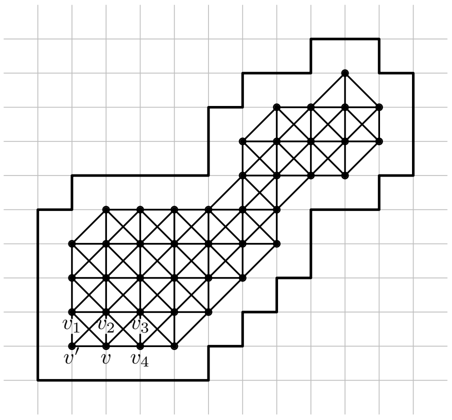

Figure 2: The intersection graph G.

The number of vertices $\widehat{N}$ of G is upper bounded by N, the number of cells in P. However, N can be far larger than $\widehat{N}$ (for example, if P is a single tail). G can be constructed in $O(n \log n + \widehat{N})$ time using a simple sweepline algorithm.

3.1 Graph Reduction

Placing a square in a corner where both edges of the polygon P have length at least 2 is known to be optimal (for example, placing the square represented by v' in Fig. 2). We generalize this result, by noting that it does not depend on the edges of P, but rather on the sets of neighbors of the vertices of G. Each of the neighbors of v' excludes more vertices from the maximum independent set than v' does.

We define the *conflict set* of a vertex v, denoted by $C(v)$, as all neighbors of v and v itself:

$$C(v) = \{u \mid (u, v) \in E\} \cup \{v\}$$

The conflict set of a vertex changes when one of its neighbors is removed from G. A vertex v is called *removable* when there exists a vertex v', a neighbor of v, such that $C(v') \subseteq C(v)$. In other words, the conflict set of v is a superset of the conflict set of v'. In Fig. 2, the conflict set of vertex v consists of v_1 to v_4, v', and v itself. The conflict set of v' consists of v_1, v_2, v, and v'. Since $C(v') \subseteq C(v)$, v is removable.

Lemma 1 *Removing a removable vertex v does not affect optimality.*

We refer to removing a removable vertex as the *superset rule*. Each vertex has degree at most 8, so we can check in constant time whether a vertex is removable. Since removing vertices from G changes the conflict set of neighboring vertices, vertices that were not removable at first can become removable later.

Next, we take a closer look at which vertices are affected when a removable vertex is removed. We already saw that only the neighbors of a removed vertex and their neighbors can become removable. A removable vertex v' remains removable when a removable vertex v is removed, unless $C(v') = C(v)$.

Lemma 2 *Given two removable vertices v and v' with $C(v') \neq C(v)$, v' remains removable after the removal of v.*

To efficiently remove all removable vertices, we maintain them in a doubly-linked list L. A vertex can be marked as removable and it stores a pointer to its location in L. At each step, we remove the first vertex in L from G and update its neighbors and the neighbors of its neighbors: unmarked vertices that become removable are added to L and marked vertices that become not removable are removed from L. This way, all removable vertices can be removed in $O(\widehat{N})$ time.

3.2 Cycles

Next, we discuss cycles of degree 2 vertices which might also contains some higher degree vertices. If a cycle C consists only of degree 2 vertices (i.e. it is not connected to the rest of G), the optimal solution contains exactly $\lfloor |C|/2 \rfloor$ vertices. For the optimality of the solution it does not matter which vertices we pick, as long as no two vertices are neighbors.

If a cycle C is connected to the rest of the graph $G \backslash C$, we can deal only with two special cases (see Fig. 3): (a) $G \setminus C$ is connected to a single vertex of C and (b) $G \setminus C$ is connected to two vertices go C, forming a triangle. If $G \backslash C$ is connected to only one vertex of C, we can remove that vertex, since for any pair of neighbors we can pick only one vertex. Note that there may be multiple edges between $G \backslash C$ and v, as long as no other vertices of C are connected to $G \setminus C$. Such single connecting vertices can also be avoided when there are multiple such vertices in C, as long as there is an odd number of vertices of C between them.

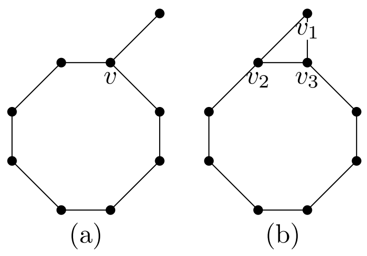

Figure 3: Cycles that can be removed.

Next, we look at connections that form triangles. In Fig. 3 (b) vertices v_1, v_2, and v_3 form a triangle and vertices v_2 and v_3 are part of the cycle C. If C has odd length, at some point in C, there must be two adjacent vertices that are both not part of the solution. In this case, these vertices can be chosen to be v_2 and v_3, without losing optimality.

If C has even length, we cannot apply this method, since there are no two adjacent vertices that are both not part of the solution. We note, however, that since vertices v_1, v_2, and v_3 form a triangle, picking any of them excludes the other two from the solution. Now, assume we pick vertex v_1 to be part of the solution. This excludes both v_2 and v_3. Since C has even length, there exists an optimal solution such that one of the neighbors of v_2 and v_3 is not used. Without loss of generality, we assume that the remaining neighbor of v_2 (i.e. not v_1 or v_3) is not used. Now vertex v_1 can be replaced by vertex v_2, without violating the property that no two adjacent vertices are part of the solution. Thus, vertex v_1 can be removed without affecting the optimality of the solution. Moreover, since v_1 may be connected to other vertices in G, not picking it is potentially better.

3.3 Diamonds

A *diamond* is a rectangle having one cell removed from each of its corners. Its corresponding graph is a rectangular graph having a vertex removed from each corner (see Fig. 4). If the height of a diamond is odd, the diamond can be removed using the superset rule and the rules for removing cycles.

We now look at a special case: a diamond consisting of four vertices having other parts of the graph attached to two opposite vertices. This configuration is shown in Fig. 5: vertices v_2, v_3, v_5, and v_6 form a diamond and the other parts of the graph are attached to two opposite vertices v_2 and v_6. The rule states that v_3 and v_5 can always be picked without affecting optimality. In other words, v_2, v_4, and v_6 can always be removed. For this rule to be applicable v_4 may be present, but this is not required. Also, there may be other vertices connected to v_2 to v_6, as long as they are not connected to v_3, v_4, and v_5. It is also possible that there are only vertices connected to v_2 (or only to v_6).

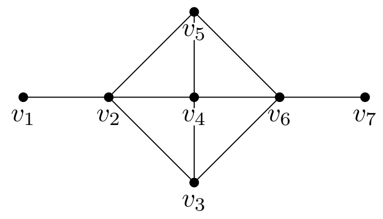

Figure 5: The configuration of a diamond.

Lemma 3 *Picking vertices v_3 and v_5 from a diamond is optimal.*

Note that we do not require that there is only one vertex on the chains between v_2 and v_6. We require only that there is an odd number of vertices on these chains. Hence, we can generalize the rule to include every diamond that satisfies this property.

3.4 Remarks

The reduction rules described in this section require that specific configurations are found in the graph G. All configurations can be found in $O(\widehat{N})$ time. Since resolving any configuration removes vertices, these rules can be applied at most a linear (in $\widehat{N}$) number of times, hence applying all rules (including the superset rule) until no rule is applicable takes $O(\widehat{N}^2)$ time.

Finally, note that no rule uses the fact that the polygon P is simple. Hence all reduction rules can also be applied to graphs constructed from polygons with holes.

4 Polygons Solvable by Graph Reduction

In the previous section, we described a number of reduction rules. However, we did not relate these rules directly to polygons. In this section we characterize the polygons that can be solved using these rules.

To construct a solvable polygon, we start out with one of four classes of *initial polygons*: The first class is the class of staircases, pyramids, and Manhattan skyline polygons. The second class consists of polygons constructed by stacking the blocks shown in Fig. 6 (see Fig. 7 (a)). The blocks may be mirrored horizontally.

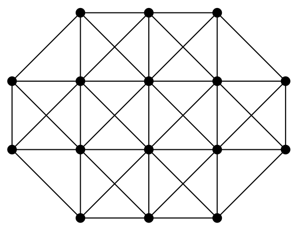

Figure 4: The graph of a diamond.

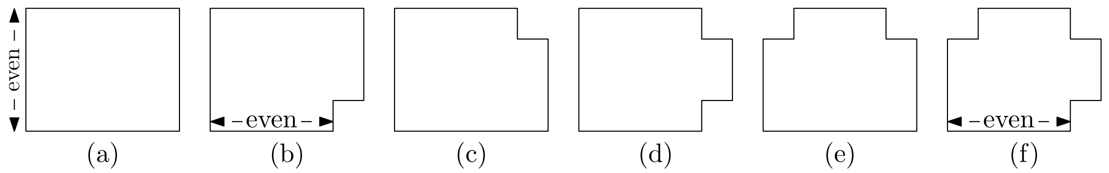

Figure 6: The various blocks. Each block starts out as an even height rectangle. Blocks (b) and (f) are required to have odd width. (a) A rectangle. (b)–(f) A single cell is removed from: (b) the lower right corner, (c) the upper right corner, (d) the upper and lower right corners, (e) the upper left and right corners, (f) the upper and lower right corners and an odd height column is removed from the upper left corner.

Note that it is allowed to stack multiple blocks next to each other on top of a wider block and that it is allowed to stack a wider block on top of multiple blocks. The third class consists of diamonds: rectangles having a single cell removed from each corner. The final class is the class of polygons constructed by starting with an arbitrary rectangle and attaching rectangles to its corners (see Fig. 7 (b)). Note that the attached rectangles may not cover an entire side of a rectangle and no two rectangles may be attached to the same corner.

Initial polygons of even height are *non-cascading polygons*. A polygon is called non-cascading if its optimal placement can be constructed regardless of the polygons that are connected to it. In other words, its optimal solution does not influence and is not influenced by the polygons connected to it. We have to restrict the type of possible connections for some non-cascading initial polygons in order for them to remain non-cascading. In the case of non-cascading staircases, pyramids, and Manhattan skyline polygons, a polygon may be attached to any horizontal edge whose height (measured from the base) is even. For non-cascading diamonds, the horizontal edge to which the other polygon is connected may not be covered entirely unless this edge has length 2.

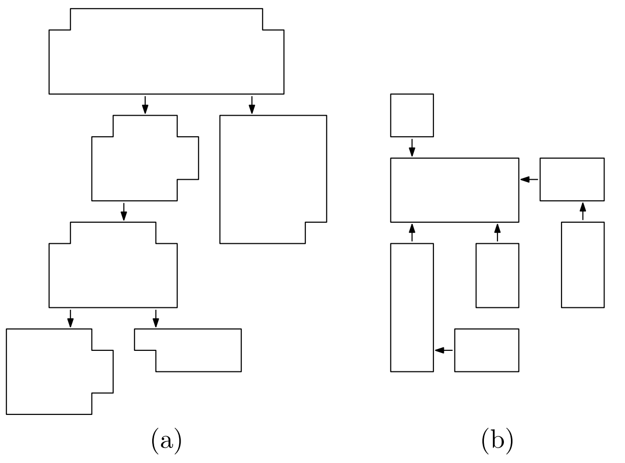

Figure 7: Constructing a polygon by: (a) stacking blocks, (b) attaching rectangles to the corners of rectangles.

Non-cascading polygons can be attached to any constructible polygon.

Rectangles of even height and width are special case of non-cascading polygons. Such rectangles can be attached anywhere. In particular, they can be attached to corners (see Fig. 8). The height and width of the rectangle need not be greater than the length of the edges of the corner, hence any corner can be extended this way.

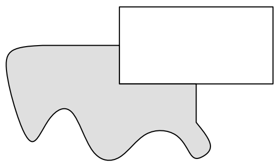

Figure 8: A corner extension.

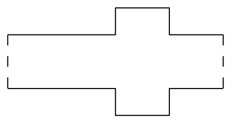

Figure 9: A universal connector polygon.

We define an *universal connector polygon* to be a polygon that can be solved without affecting the connected polygons in any way. Any two polygons may be connected by means of a universal connector polygon. We distinguish two types of universal connector polygons. The first type is the tight corridor described in Section 2. Two polygons connected by a tight corridor correspond to two disconnected graphs that can be solved individually. The second type is a variation on the diamond: an even width rectangle of height 4 having even width rows of cells removed from each corner. The rectangles that are removed from the left corners must have the same width. The same must hold for the rectangles that are removed from the right corners (see Fig. 9). Universal connector polygons can be extended by attaching non-cascading polygons to any edge or by attaching even height and width rectangles to corners.

Any polygon can be extended by attaching tails to it (see Section 2).

Theorem 4 *Every polygon constructed by the construction scheme of Section 4 can be solved using the graph reduction rules.*

The class of polygons constructible using the above construction scheme is significantly larger than the class

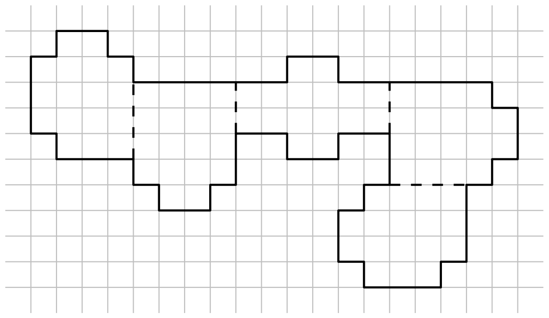

Figure 10: A polygon constructed using the construction scheme. Dashed lines represent the borders of the building blocks.

of previously solvable polygons. An example polygon is shown in Fig. 10.

5 Polygons Solvable in Quadratic Time

In the previous section, we presented a construction scheme for polygons solvable in time polynomial in $\widehat{N}$ (the number of possible square locations) using the graph reduction method. In this section, we present a different class of polygons which does not require the construction of the intersection graph; the constructed polygons can be solved in $O(n^2)$ time. Though this new class has some overlap with the class of the previous section, neither is contained in the other.

The new construction scheme is very similar to the previous scheme. There are, however, some differences. Diamonds are replaced by diamonds having an arbitrary number of steps and the non-cascading diamonds are replaced by even height diamonds having an arbitrary number of steps. Furthermore, the extension of rectangles placed in corners has an additional requirement: the height and width of the rectangle may not be equal to the length of the edges of the corner.

To solve these polygons, we find specific configurations of edges in the polygon. These configurations will correspond to (parts of) the polygons constructed using the construction scheme. Since we also need to know which edge of the polygon is closest to another edge, we first define the distance between two horizontal edges (two vertical edges are treated analogously). Since we do not need the distance between a horizontal and a vertical edge, we define this to be infinite.

We define the *slab* of an edge e, denoted by slab(e), as the region bounded by e and two half-lines orthogonal to e starting at the two endpoints of e, such that the interior of the polygon intersects slab(e) in the immediate neighborhood of e. A slab contains an edge f if one of the endpoints of f lies in the interior or on the boundary of the slab or f intersects the slab.

Given two horizontal edges e and f of a rectilinear grid polygon, the distance between these two edges is defined to be infinite when slab(e) does not contain f and slab(f) does not contain e, or when the two edges are connected by means of a single vertical edge. Otherwise, the distance is the difference between their respective y-coordinates.

We now define the edge e closest to another edge f as the edge that has the smallest distance to edge f. An edge can have multiple closest edges: if multiple edges have the smallest distance to an edge e, they are all part of the set of closest edges of e. The closest edges are needed to efficiently check whether we need to split the polygon during the removal of configurations.

The configurations are shown in Fig. 11. Here we describe two of these configurations in detail and analyze the number of squares that can be placed when removing them. Full details can be found in [8]. Some configurations can be solved by using other configurations in combination with tail removal. These configurations are shown to make it easier to see the correlation between the construction scheme and the configurations. When describing the configurations, we say that certain rectangles do not contain edges in their interior. Here a rectangle contains an edge if (a part of) the edge lies in the interior of the rectangle. Edges lying on the boundary of the rectangle are allowed. All configurations may be mirrored and rotated.

Configuration (a): A rectangular configuration C. The rectangle defined by the horizontal edge and the shortest vertical edge does not contain any edges of the polygon. The height h of C is the length of the shortest of the two vertical edges and the width w of C is the length of the horizontal edge. The height h needs to be strictly greater than 1. When C is removed, we add $\lfloor h/2 \rfloor \cdot \lfloor w/2 \rfloor$ squares. Note that if the shortest vertical edge has odd length, C is not removed completely: it is reduced to a single row.

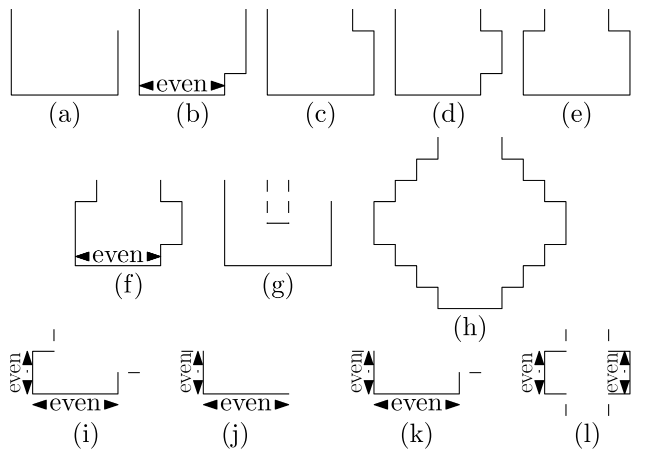

Figure 11: The configurations.

Configuration (l): A universal connector configuration C. The two vertical edges have the same even length l and the y-coordinates of their endpoints are the same. The distance between the two edges is 4. Both edges are connected to two edges of length 1 that are directed towards the opposite edge. The rectangle defined by the two vertical edges does not contain any edges of the polygon. When C is removed, we add l squares.

Next, we sketch how to use the configurations shown in Fig. 11 to solve the polygons constructed by the construction scheme. Staircases, pyramids, and Manhattan skyline polygons can be solved by repeatedly using configurations (a) and (g) on the base and the two vertical edges connected to it. The blocks shown in Fig. 6 can be solved by using the corresponding configurations: block (a) can be solved using configuration (a), block (b) using configuration (b), and so on. Diamonds having an arbitrary number of steps can be solved by using configuration (h). Polygons constructed by starting with an arbitrary rectangle and attaching rectangles to its corners can be solved by repeatedly using configuration (a) in combination with tail removal. For details see [8].

Since non-cascading polygons are special cases of the above polygons, the configurations used to solve them are the same. The even height and width rectangles that can be placed in corners can be solved using configurations (i), (j), and (k), depending on whether the edges of the corner are longer or shorter than the edges of the rectangle.

The tight corridors that can be used as universal connectors will be removed before the algorithm is applied to the polygon. Tight corridors formed during the removal of configurations will be removed as soon as they are formed. The diamonds that are used as universal connector polygons can be solved using configuration (l). Finally, tails will also be removed before and while the algorithm is applied to the polygon.

Solving the polygons constructed by this scheme now becomes quite simple: we find one of the configurations, fill it, and continue with the remainder of the polygon. To keep the algorithm this simple, we need to ensure that the remaining part of the polygon is described by its edges. Hence we remove all tight corridors (causing the polygon to be split) and tails after each step.

Lemma 5 *The polygon can be split at most $n/4 - 1$ times.*

Lemma 6 *A polygon can be split in $O(n)$ time, while updating the closest edges.*

Lemma 7 *While solving the polygon, configurations that do not reduce the complexity of the polygon are used at most a linear number of times.*

From Lemmas 5, 6, and 7 it follows that the configuration removal algorithm runs in $O(n^2)$ time.

Theorem 8 *Every polygon constructed by the construction scheme of Section 5 can be solved in $O(n^2)$ time using the configuration removal algorithm.*

6 Conclusion and Open Problems

We described a number of techniques that can be used to solve certain instances of the 2×2 simple packing problem on simple polygons. Our methods significantly extend the class of polygons for which the 2×2 simple packing problem is solvable in polynomial time. The graph reduction technique is polynomial in $\widehat{N}$ and the configuration removal technique runs in $O(n^2)$ time. Both techniques return the optimal number of squares for polygons constructed using their respective construction schemes.

The complexity status of the 2×2 simple packing problem remains open. Nevertheless, it is an interesting open question if a PTAS that runs in time polynomial in n (not just polynomial in $\widehat{N}$) exists. Another challenging problem is to find an exact algorithm for classes of polygons that do not require construction schemes to describe them, such as the class of rectilinear convex simple polygons.

Acknowledgements. The authors would like to thank Prosenjit Bose for helpful comments.

References

[1] F. Berman, D. Johnson, T. Leighton, P. Shor, and L. Snyder. Generalized planar matching. *Journal of Algorithms*, 11(2):153–184, 1990.

[2] F. Berman, F. Leighton, and L. Snyder. Optimal tile salvage. Technical Report ADA119117, Purdue University, May 1982.

[3] T. Chan. A note on maximum independent sets in rectangle intersection graphs. *Information Processing Letters*, 89(1):19–23, 2004.

[4] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke. The open problem project. `http://maven.smith.edu/~orourke/TOPP/`.

[5] M. Dulieu, D. El-Khechen, J. Iacono, and N. van Omme. Packing 2×2 unit squares into grid polygons is NP-complete. In *Proc. 21st Canadian Conference on Computational Geometry*, pages 33–36, August 2009.

[6] D. El-Khechen. *Decomposing and Packing Polygons.* PhD thesis, Concordia University, April 2009.

[7] D. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.

[8] A. van Renssen. The 2×2 simple packing problem. Master's thesis, Technische Universiteit Eindhoven, 2010.