Computing Heesch Numbers of Polykites

Gulinazi Julati[1], Sameer Narendran[1], Lucas Ortengren[1], and Nathan Sullivan[1]

[1]Madison Experimental Mathematics Lab, University of Wisconsin Madison

December 2023

1 Background

Our project began as an investigation into the recently discovered aperiodic monotile known as the hat, pictured in Figure 1.

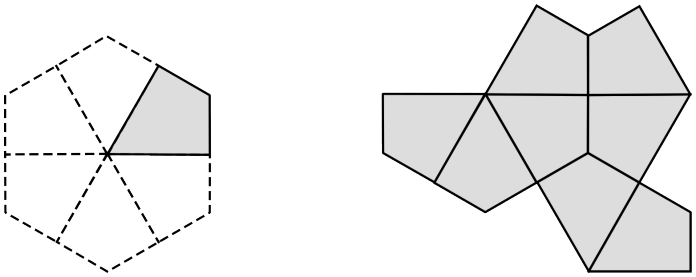

Figure 1: Kite relative to a regular hexagon (left) and the hat polykite (right).

A tiling is *aperiodic* if there are no translational symmetries, and a set of tiles is aperiodic if it only admits aperiodic tilings. The two Penrose tiles have long been known to admit aperiodic tilings, but finding a single tile that could produce an aperiodic tiling had been an open problem for decades, until a solution was published earlier this year. David Smith, a hobbyist mathematician, discovered such a tile, dubbed the "hat" while creating shapes in the software Polyform Puzzle Solver. It is constructed from kites, which are made by splitting a regular hexagon into six equal parts; connecting eight of these kites yields the hat [1].

2 Heesch's Problem

While researching the aperiodic monotile, and tiles in general, we came across an open question in tiling known as Heesch's problem. First, we define the *corona* of a tile to be all the tiles that surround it. The maximal number of coronas a shape can have is its *Heesch*

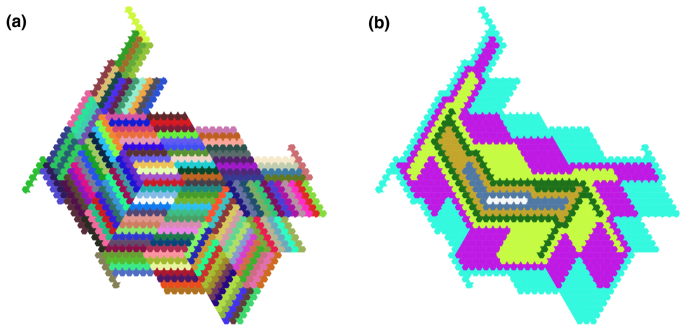

Figure 2: A patch made from a tile Heesch number 6 [2].

number. A nontiler, such as a circular disk, has a Heesch number of zero. A tiler, such as a square, has a Heesch number of infinity. Heesch's problem is concerned with finding shapes that have Heesch numbers of neither zero nor infinity. The Heesch number is conjectured to be unbounded, but the highest known Heesch number is only 6, discovered by Bojan Bašić in 2020.

There are two common definitions of a Heesch number: tilings where holes are permitted in the outer corona and tilings where the holes are not permitted. In our work, we decided to focus solely on the former problem to simplify the computational complexity. However, note that if the Heesch number of a shape permitting holes in the outer corona is $H = h$, then the Heesch number without holes in the outer corona is either h or $h - 1$.

While researching Heesch numbers, we were greatly inspired by a research paper on polyominoes, polyiamonds, and polyhexes. The author, Craig Kaplan, was able to create a program to reduce the search for a shape's Heesch number to a Boolean satisfiability problem [3]. Because the recently discovered aperiodic monotile is a polykite, we thought there might be interesting results to be found from calculating the Heesch numbers of polykites.

3 Implementation

Our work began with recreating a program capable of generating Boolean clauses from a given tile, as detailed in [3]. The idea is to translate the difficult geometric problem into one able to be solved using existing tools. We accomplish this by generating a logical statement that is satisfiable if and only if the tile can be made to form an n-corona without overlaps or gaps. This statement can then be input into a SAT solver, a program designed to determine the satisfiability of Boolean strings. This method takes advantage of the extensive optimization of SAT solvers, offloading much of the hard work to preexisting software.

Before computing the Heesch numbers of polykites, there are two important steps to take: generating all polykites of size n and removing all tiling polykites. Generating polykites is nontrivial, but the most important step is identifying which tiles *do not* tile the plane. A shape that tiles the plane is defined to have a Heesch number of infinity, and if we ran our

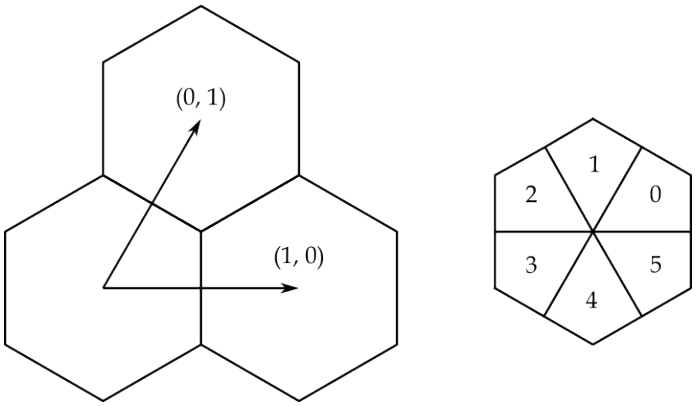

Figure 3: A grid of hexagons, and a grid for kites.

program on such a shape it would not terminate. Luckily for us, Joseph Myers, a mathematician and tiling enthusiast, graciously provided a complete list of non-tiling polyforms that he had previously computed.

The algorithm we created is a programmatic reduction from the Heesch problem for an arbitrary polykite to a Boolean satisfiability problem. Consider the problem of whether a shape S has a Heesch number at least k. By modeling this problem as a series of constraints on where to place copies of S and creating the corresponding variables, we can use a SAT solver to determine if there is a configuration of the tiles to create k coronas around the original shape.

3.1 A Polykite Grid

To represent the polykites, we must first define a grid: We used a three-coordinate system. First, we considered an infinite grid of hexagons. The x-coordinate goes right and left along the main row, and the y-coordinate goes up and to the right if positive, or down and to the right if negative. For the z-coordinate, each hexagon is split into six sections, as shown above.

Using this grid, each polykite has 6 possible rotations and 6 reflected rotations, totaling 12 possible transformations. We can perform the rotations using the matrices R and the reflections using S:

$$R = \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}, \quad S = \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix}$$

3.2 Designing an Algorithm

We first generated a finite kite-grid as described above with its size determined by the size of the input shape S. We can then create a variable for each cell in our finite grid, c_i, which is true if and only if the cell is occupied. In addition, we must create a variable for every possible transformation in the grid, t_i, where t_i is true if and only if the transformation is used. We designate t_0 as the identity transformation. To do this, we pre-compute

the transformations of every possible rotation and translation of S for each corona. These transformations can be uniquely identified by the tuple $(k\ x\ y\ r)$ where k is the corona, x, y are the x and y-translations respectively, and r is the rotation and reflection. Out of the transformations generated, any transformation in corona i that could not be adjacent to a transformation in corona $i-1$ was immediately discarded. The halo of each transformation, the set of grid cells surrounding it, was pre-computed as well.

Similar to the notation Kaplan uses in his paper [3], we will denote the set of transformations that can be used in the k-th corona as T_k. In addition, we will also use the notation $HALO(c_i), HALO(t_i)$ to describe the halo of a cell or a transform and the notation $CELLS(t_i)$ to describe the cells occupied by a transform. In order to apply a SAT solver, the boolean formula must be in conjunctive normal form: a conjunction of clauses, each of which must be composed of a disjunction of boolean literals.

Then, we can construct the following clauses in conjunctive normal form that describe which subset of transforms create a valid configuration for k coronas:

- The identity transformation must always be used:

$$t_0$$

- If a transformation is used, all of the cells it covers must also be used:

$$t_i \Rightarrow c_j \equiv \neg t_i \vee c_j, \quad \forall c_j \in CELLS(t_i), \forall t_i$$

- If a cell is used, then there must be a transformation covering that cell that is used:

$$c_j \Rightarrow t_1 \ldots t_n \equiv \neg c_j \vee t_1 \vee \ldots \vee t_n \quad \forall t_i \text{ where } c_j \in CELLS(t_i), \forall c_j$$

- If a transformation is used and it is in an inner corona (a corona between 0 and $k-1$), then all of the cells adjacent to that transformation must also be used:

$$t_i \Rightarrow c_j \equiv \neg t_i \vee c_j, \quad \forall c_j \in HALO(t_i), \forall t_i \in T_1, \ldots, T_{k-1}$$

- If a transformation is used, then any transformation that is overlapping with it cannot be used:

$$t_i \Rightarrow \neg t_j \equiv \neg t_i \vee \neg t_j, \quad \forall t_i, t_j, i \neq j, CELLS(t_i) \cap CELLS(t_j) \neq \emptyset$$

- If a transformation is used in the i-th corona, then it must be adjacent to a transformation in the $(i-1)$-th corona:

$$t \Rightarrow t_1 \vee \ldots \vee t_n \equiv \neg t \vee t_1 \vee \ldots \vee t_n, \quad t \in T_i, t_1, \ldots, t_n \in T_0, \ldots, T_{i-1}, \\ CELLS(t) \cap CELLS(t_j) \neq \emptyset, \forall 1 \leq j \leq n$$

- If a transformation is used in the i-th corona, then it cannot be adjacent to any transformation in the $0, \ldots, (i-2)$-th coronas:

$$t_i \Rightarrow \neg t_j \equiv \neg t_i \vee \neg t_j, \quad \forall t_i \in T_i, t_j \in T_j, j < i-1, \\ CELLS(t_i) \cap HALO(t_j) \neq \emptyset$$

After finding a satisfying configuration of variables for the boolean formula, we can reconstruct the k coronas by seeing which transformations were used and graph the output. Using this method to check whether a shape S has a Heesch number of at least k, we can incrementally increase k until we encounter a boolean formula that is unsatisfiable.

4 Results

Our program was run on Joseph Myers' list of all non-tiling polykites [4] up to and including 9-kites to determine their Heesch Numbers.

Table 1: Heesch Number for n-kites until $n = 9$

Shape Size	H = 0	H = 1	H = 2	Total Non-Tilers
2	1	0	0	1
4	4	1	0	5
5	10	16	0	26
6	10	3	0	13
7	152	48	3	203
8	628	168	9	805
9	1869	152	0	2021

Table 1 shows the distribution of Heesh numbers (permitting holes in the outer corona) of all non-tiling polykites up to and including 9-kites.

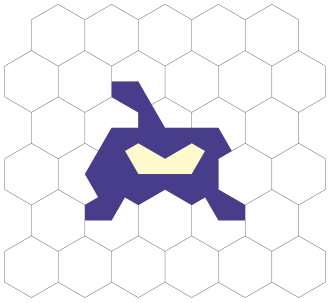

Figure 4: A 4-kite with Heesch number 1

The 4-kite depicted above represents the smallest polykite exhibiting a nontrivial Heesch number.

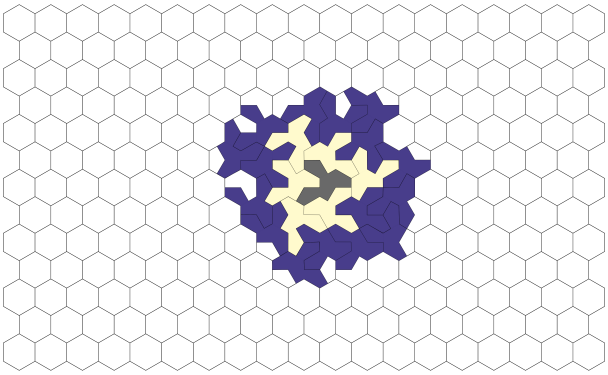

Figure 5: An 8-kite with Heesch number 2

The 8-kite depicted above exhibits a nontrivial Heesch number of 2. Our algorithm was executed on all polykites up to size 9, yet no shapes with a Heesch number exceeding two were discovered.

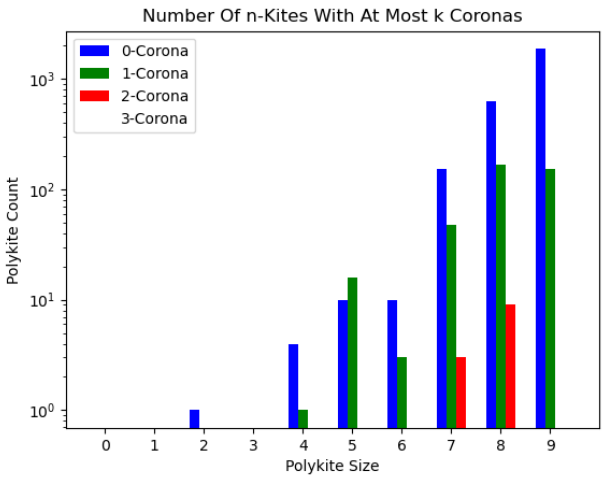

Figure 6: Number of n-kites with at most k coronas

4.1 Performance

Our program [5] was written in Python and was run on an Intel i7-10510U processor with 8 cores. When run on polykites of sizes $1, \ldots, 9$, the program took on average 2.2 seconds to determine if a 1-corona existed and 9.8 seconds on average to determine if a 2-corona existed. The program utilized the Python library PySAT [6] to find satisfying solutions to the boolean formula.

5 Conclusion and Future Direction

As part of our future work, we could improve the software to disallow holes in the outermost corona. Our software also still has room for improvement in terms of efficiency. These enhancements would enable us to effortlessly compute the Heesch numbers for every n-kite up to n = 17. Additionally, our software has the flexibility to be adapted for the analysis of other polyforms derived from alternative uniform tilings.

One question that has piqued the curiosity of our team members is whether there exist polykites with a Heesch number exceeding 2. Other possibilities to explore in the future include extending the software to other combinations of regular shapes and Heesch numbers of regular polyform in higher dimensions.

Acknowledgements

We want to extend our appreciation to Joseph Myers for sharing his collection of non-tiling polykites with us. Additionally, we wish to convey our thanks to the Madison Experimental Mathematics lab, as well as to Caglar Uyanik, Grace Work, and Karan Srivastava. We also want to give a special thanks to Robert Argus and Albert Ai for their invaluable guidance throughout the project

References

[1] David Smith et al. *An Aperiodic Monotile*. May 2023. DOI: 10.48550/arXiv.2303.10798. eprint: 2303.10798 (cs, math). (Visited on 09/24/2023).

[2] Bojan Bašić. "A Figure with Heesch Number 6: Pushing a Two-Decade-Old Boundary". In: *The Mathematical Intelligencer* 43.3 (Sept. 2021), pp. 50–53. ISSN: 1866-7414. DOI: 10.1007/s00283-020-10034-w. (Visited on 11/06/2023).

[3] Craig S. Kaplan. "Heesch Numbers of Unmarked Polyforms". In: (Dec. 2022). DOI: 10.11575/cdm.v17i2. eprint: 2105.09438 (cs). (Visited on 09/29/2023).

[4] Joseph Myers. *Polyform Tiling*. URL: https://www.polyomino.org.uk/mathematics/polyform-tiling/ (visited on 12/22/2023).

[5] Sameer Narendran, Lucas Ortengren, and Gulinazi Julati. *Sameer-N012/Mxm-Aperiodic-Monotiles*. Nov. 26, 2023. URL: https://github.com/sameer-n012/mxm-aperiodic-monotiles (visited on 12/22/2023).

[6] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. "PySAT: A Python Toolkit for Prototyping with SAT Oracles". In: *SAT*. 2018, pp. 428–437. DOI: 10.1007/978-3-319-94144-8_26. URL: https://doi.org/10.1007/978-3-319-94144-8_26.