EuroComb 2005 DMTCS proc. **AE**, 2005, 369–374

Counting Polyominoes on Twisted Cylinders

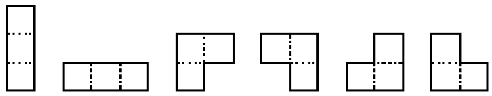
Gill Barequet[1], Micha Moffie[1], Ares Ribó[2†] and Günter Rote[2]

[1] *Department of Computer Science, Technion—Israel Institute of Technology, Haifa 32000, Israel*
[2] *Freie Universität Berlin, Institut für Informatik, Takustraße 9, 14195 Berlin, Germany*

We improve the lower bounds on Klarner's constant, which describes the exponential growth rate of the number of polyominoes (connected subsets of grid squares) with a given number of squares. We achieve this by analyzing polyominoes on a different surface, a so-called *twisted cylinder* by the transfer matrix method. A bijective representation of the "states" of partial solutions is crucial for allowing a compact representation of the successive iteration vectors for the transfer matrix method.

1 Introduction

A polyomino of size n, also called an n-omino, is a connected set of n adjacent squares on a regular square lattice. (Connectivity is through edges only). *Fixed* polyominoes are considered distinct if they have different shapes *or* orientations. The six fixed *triominoes*—polyominoes of size 3 are shown on the side.



Counting polyominoes has received a lot of attention in the literature, see Barequet et al. [1] for an overview. The number $A(n)$ of fixed polyominoes of size n is known to be exponential in n. Klarner [6] showed that $A(n) \sim C\lambda^n n^\theta$ for some constants $C > 0$ and $\theta \approx -1$. The limit $\lambda := \lim_{n\to\infty} A(n+1)/A(n)$ is commonly called *Klarner's constant*. The best-known published lower and upper bounds are 3.927378 [5] and 4.649551. However, as pointed out in [2], this lower bound is based on an incorrect assumption, and should have been corrected to 3.87565.

The most successful approach for computing values of $A(n)$ is Jensen's transfer matrix algorithm [4, 5], which is based on the algorithm of Conway and Guttmann [3]. The transfer matrix approach computes "partial polyominoes" of larger and larger sizes in a dynamic-programming manner. For each "state" of a partial solution, and for each number of cells, one has to maintain the number of partial polyominoes with that state. Thus, the algorithm has to handle a large table whose entries are indexed by states. The success of Jensen's approach is based on treating only a small fraction of all possible states. This so-called *pruning* of states requires the algorithms to encode and store states explicitly, using a hash table. Jensen [5] could obtain the values up to $A(56)$, using parallel computers.

In this paper we count polyominoes on a different grid structure, a twisted cylinder. A *twisted cylinder* of width W is obtained from the integer grid $\mathbb{Z} \times \mathbb{Z}$ by identifying point (i, j) with $(i + 1, j + W)$, for all i, j. Geometrically, it can be imagined to be an infinite tube; see Figure 1.

†Supported by the Deutsche Forschungsgemeinschaft within the European Graduate Program *Combinatorics, Geometry and Computation* (No. **GRK** 588/2).

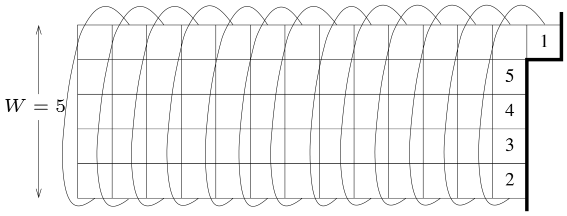


Fig. 1: A twisted cylinder of width 5. The wrap-around connections are indicated; for example, cells 1 and 2 are adjacent.

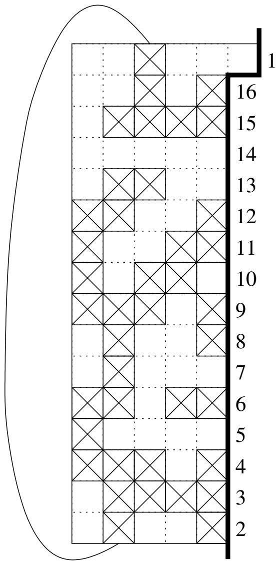
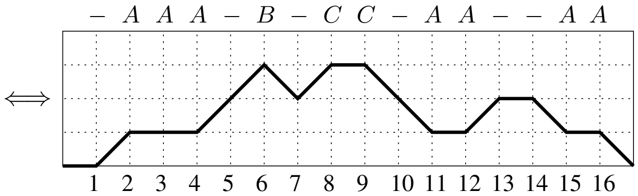



Fig. 2: Left: A snapshot during the construction of a polyomino. Everything to the left of the solid *boundary line* is already determined. This state has the connected components $\{2, 3, 4, 11, 12, 15, 16\}$, $\{6\}$, $\{8, 9\}$. Note that the bottom cell of the second column is adjacent to the top cell of the third column. The numbers are the labels of the boundary cells. Right: the same state encoded by assigning a distinct letter to each component and using the symbol "$-$" for empty cells, and the corresponding Motzkin path $(0, 1, 0, 0, 1, 1, -1, 1, 0, -1, -1, 0, 1, 0, -1, 0, -1)$.

The twist allows us to build up the cylinder incrementally, one cell at a time, through a *uniform* process. This leads to a simpler recursion. The states have a very clean combinatorial structure, and they are in bijection with so-called Motzkin paths. We use this bijection as a space-efficient scheme for addressing the entries of the state vector in the computer.

We could improve the lower bound on Klarner's constant to 3.980137. The full paper [2] contains all details and proofs which are omitted in this extended abstract.

2 The Transfer Matrix Algorithm

The strategy is as follows. The polyominoes are built from left to right, adding one cell of the twisted cylinder at a time. For each partial polyomino that is built in this way, we have to record which of the W *boundary cells* are occupied, and how the occupied cells are connected to each other, see Figure 2. This information, which forms the *state* of a (yet incomplete) polyomino, is therefore a partition of the set $\{1, .., n\}$, with the following additional properties: (1) If two adjacent cells i and $i+1$ are occupied, they must belong to the same component. (2) Different components must be *non-crossing*: For $i < j < k < l$, it is impossible that i and k belong to one component and j and l belong to a different component.

2.1 Motzkin Paths

For a compact representation and indexing of states in the computer, we use Motzkin paths. This representation has also been applied by Barequet and Moffie [1]. A *Motzkin path* of length n is a path from

$(0,0)$ to $(n,0)$ in a $n \times n$ grid, consisting of up-steps $(1,1)$, down-steps $(1,-1)$, and horizontal steps $(1,0)$, that never goes below the x-axis. A Motzkin path of length n is represented as a string of n symbols of the alphabet $\{1,-1,0\}$. The number M_n of Motzkin paths of length n is obviously bounded by $M_n \le 3^n$, which is not a too crude bound. These *Motzkin numbers* M_n have many other combinatorial interpretations [9].

Theorem 1 *There is a bijection between states of length W and Motzkin paths of length $W+1$.*

Proof: (Sketch.) Consider the edges between adjacent boundary cells. An edge between two occupied cells or between two empty cells is mapped to the horizontal step 0. A transition between an occupied cell and a free cell is encoded as follows; see Figure 2 for an example.

- The left edge of the first block of adjacent occupied cells within a connected component is mapped to 1.
- The left edge of each remaining block is mapped to -1.
- The right edge of the last block of a component is mapped to -1.
- The right edge of each remaining block is mapped to 1.

In other words, when a new component starts, the path will rise to a new level $(+1)$. All cells of the component will lie on this level. When the component is interrupted, the path rises to a higher level $(+1)$, essentially pushing the current block on a stack, to be continued later. When the component resumes, the path will come down to the correct level (-1). At the very end of the component, the path will be lowered to the level that it had before the component was started (-1). For empty cells, the path lies on an even level, whereas occupied cells lie on odd levels. A connected component which is nested within k other components lies on level $1+2k$.

The conversion from a Motzkin path to a partition of boundary cells can be easily done by scanning the path from left to right, maintaining a stack of partially finished components. □

There other possible ways to encode the states. Jensen [4] used signature strings of length W containing the five digits 0–4, which Knuth [7] replaced by the more intuitive five-character alphabet $\{$`0`, `1`, `(`, `)`, `-`$\}$.

3 Polyominoes on a Twisted Cylinder

Lemma 1 *For the number $Z_W(n)$ of polyominoes of size n on a twisted cylinder of width W, we have $Z_W(n) \le A(n)$.*

Proof: For an n-omino Y on the cylinder we can select a spanning tree T of the dual graph of Y; keeping the adjacent cells of T connected defines an n-omino in the plane. The inverse process of wrapping the plane onto the cylinder always leads back to the same Y. □

Thus, Klarner's constant λ, which is the growth rate of $A(n)$, is lower bounded by the growth rate λ_W of $Z_W(n)$, that is:

$$\lambda \ge \lambda_W = \lim_{n\to\infty} \frac{Z_W(n+1)}{Z_W(n)}$$

3.1 Successor states

Let $\mathcal{S}$ be the set of states. For each state $s \in \mathcal{S}$, we have two possible *successor states* $succ_0(s)$ and $succ_1(s)$: We add a new cell adjacent to cells 1 and W. This cell can be either empty or occupied. When computing the successor state, one has to take care of the changes in the connectivity of the components, and one has to take into account that the cells $1, .., W-1$ are renumbered to $2, .., W$. The old cell W becomes disconnected from the boundary, and the new cell becomes cell 1. Essentially, the polyomino is rotated around the cylinder by one cell. The full paper [2] describes in detail how the update of states is carried out at the level of the encoding by Motzkin paths.

Note that $succ_0(s)$ does not exist if adding an empty cell causes some connected component to become isolated from the boundary, because in this case a connected polyomino could never be completed. This happens exactly when $\{W\}$ forms a singleton component in s.

Define the vector $\mathbf{x}^{(i)}$ of length $|\mathcal{S}|$ with components:

$$\mathbf{x}_s^{(i)} := \sharp\{\text{partial polyominoes with } i \text{ occupied cells in state } s\} \tag{1}$$

Then $\mathbf{x}^{(n)}_{\{\{W\}\}}$ is the number $Z_W(n)$ of n-ominoes on the twisted cylinder of width W, where $\{\{W\}\}$ denotes the state where W is the only occupied cell. We can set up the following recursion:

$$\mathbf{x}_s^{(i+1)} = \sum_{s':s=succ_0(s')} \mathbf{x}_{s'}^{(i+1)} + \sum_{s':s=succ_1(s')} \mathbf{x}_{s'}^{(i)} \qquad \forall s \in \mathcal{S} \tag{2}$$

The apparent dependence of the vector $\mathbf{x}^{(i+1)}$ on itself does not cause any problems. If the components are processed in an appropriate order, there is no cyclic dependence.

For maintaining an array whose entries are indexed by Motzkin paths, we use standard ranking and unranking algorithms for lattice paths in lexicographic order, cf. [8, Section 3.4.1]. After precomputing for each lattice point (i, j) the number of paths from (i, j) to the target point $(n, 0)$ and storing this information in an $O(n \times n)$ table, conversion between a Motzkin path and an integer between 1 and M_n can be carried out in $O(n)$ time. It turns out that this lexicographic order of states s is also appropriate for evaluating the iteration (2) to avoid using a component $\mathbf{x}_{s'}^{(i+1)}$ that has not been defined.

In matrix form, the system of equations (2) can be written in terms of two transfer matrices A and B.

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i+1)}A + \mathbf{x}^{(i)}B \tag{3}$$

A and B are two 0-1-matrices that reflect the transition from s to $succ_0(s)$ and $succ_1(s)$, respectively. With an appropriate initial vector $\mathbf{x}^{(1)}$, we can thus compute the values $Z_W(n)$ for $n = 1, 2, \ldots$. However we are mainly interested in the asymptotic growth rate $\lambda_W := \lim_{n\to\infty} Z_W(n+1)/Z_W(n)$, which is determined by the Perron-Frobenius eigenvalue of the operator $\mathbf{x}^{(i)} \mapsto \mathbf{x}^{(i+1)}$ given by (3). Actually, the matrix of this system is not irreducible because there are some states that can never be reached. However, one can argue that these "superfluous" states have no influence on the result, and the sequence of vectors $\mathbf{x}^{(i)}$ converges as predicted in the Perron-Frobenius theorem.

3.2 Backward recursion

In our program we use the transpose of the transfer matrix operator, which has of course the same Perron-Frobenius eigenvalue.

$$\mathbf{y}^{(i-1)} = A\mathbf{y}^{(i-1)} + B\mathbf{y}^{(i)}, \text{ for } i = 0, -1, -2, \ldots$$

This translates into a very simple iteration procedure

$$\mathbf{y}_s^{\mathrm{new}} = \mathbf{y}_{succ_0(s)}^{\mathrm{new}} + \mathbf{y}_{succ_1(s)}^{\mathrm{old}} \qquad \text{for all } s \in \mathcal{S} \tag{4}$$

(If $succ_0(s)$ does not exist, the corresponding value is simply omitted.) This iteration is more convenient to program than the forward iteration (2). Moreover, it interacts beneficially with memory hierarchies of modern computer hardware.

We start with the vector $\mathbf{y}$ of all ones (any vector with positive entries will do), and iterate (4) until convergence. The following lemma shows how to compute bounds for λ_W.

Lemma 2 *Let $\mathbf{y}^{\mathrm{old}}$ be any vector with positive entries and let $\mathbf{y}^{\mathrm{new}} = A\mathbf{y}^{\mathrm{new}} + B\mathbf{y}^{\mathrm{old}}$. Let λ_{low} and λ_{high} be, respectively, the minimum and maximum values of $\mathbf{y}_s^{\mathrm{new}}/\mathbf{y}_s^{\mathrm{old}}$ over all components s. Then, $\lambda_{\mathrm{low}} \le \lambda_W \le \lambda_{\mathrm{high}}$.* □

By Lemma 1, $\lambda_{\mathrm{low}} \le \lambda_W \le \lambda$, and thus, λ_{low} is a lower bound on Klarner's constant as well.

4 Results

We have written a program in C to implement the iteration. We initially compute the successors $succ_0(s)$ and $succ_1(s)$ for each state s and store them in two arrays. The iteration vectors $\mathbf{y}^{\mathrm{old}}$ and $\mathbf{y}^{\mathrm{new}}$ are single-precision floating-point vectors.

The table summarizes our results. We iterate the equations until $\lambda_{\mathrm{high}} < 1.000001\,\lambda_{\mathrm{low}}$. (We check this every ten iterations. The number of iterations is shown in the second column.) The bounds in the table are rounded conservatively.

W	#it	λ_{low}	λ_{high}
12	90	3.853547	3.853551
13	110	3.877518	3.877521
14	120	3.897315	3.897319
15	130	3.913878	3.913883
16	140	3.927895	3.927899
17	160	3.939877	3.939882
18	170	3.950210	3.950215
19	190	3.959194	3.959198
20	200	3.967059	3.967064
21	220	3.973992	3.973996
22	240	3.980137	3.980142

So the best lower bound that we obtained is $\lambda > 3.980137$, for $W = 22$. For $W \le 20$, we also wrote an independent checking routine in Maple that takes the vector of the last iteration as input and computes a lower bound by exact arithmetic. For $W = 20$, this leads to a "certified" bound of $\lambda \ge 348080/87743 > 3.96704$.

We performed the calculations on a workstation with 32 gigabytes of memory. The running time for the largest example ($W = 22$) was about 6 hours. Going to $W = 23$ or even $W = 24$ would perhaps be feasible with some optimizations and programming tricks to save memory. To break the "magical" barrier of 4 for λ one probably needs to go at least to $W = 27$ with this approach, which is by far out of reach for current hardware.

Acknowledgements

We thank Stefan Felsner for discussions about the bijection between states and Motzkin paths.

References

[1] G. Barequet and M. Moffie, The complexity of Jensen's algorithm for counting polyominoes, *Proc. 1st Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, New Orleans, ed. L. Arge, G. F. Italiano, and R. Sedgewick, SIAM, Philadelphia 2004, pp. 161–169.

[2] G. BAREQUET, M. MOFFIE, A. RIBÓ, G. ROTE, *Counting polyominoes on twisted cylinders*, manuscript, Dec. 2004, 29 pp., submitted for publication, http://www.inf.fu-berlin.de/˜rote/

[3] A. R. CONWAY AND A. J. GUTTMANN, On two-dimensional percolation, *J. Physics, A: Mathematical and General* **28** (1995), 891–904.

[4] I. JENSEN, Enumerations of lattice animals and trees, *J. of Statistical Physics*, **102** (2001), 865–881.

[5] I. JENSEN, Counting polyominoes: A parallel implementation for cluster computing, in: *Computational Science—ICCS 2003*, ed. P. M. A. Sloot et al., *Lecture Notes in Computer Science*, Vol. 2659, pp. 203–212, Springer-Verlag, 2003.

[6] D. A. KLARNER, Cell growth problems, *Canad. J. Math.* **19** (1967), 851–863.

[7] D. E. KNUTH, Programs POLYNUM and POLYSLAVE,
http://sunburn.stanford.edu/˜knuth/programs.html#polyominoes

[8] D. L. KREHER AND D. R. STINSON, *Combinatorial Algorithms, Generation, Enumeration and Search* (CAGES), CRC Press, 1998.

[9] R. P. STANLEY, *Enumerative Combinatorics*, Vol. 2, Cambridge Studies in Advanced Mathematics, 1999.