$\lambda > 4$

Gill Barequet* Günter Rote† Mira Shalah*

Abstract

A *polyomino* (or "animal") is an edge-connected set of squares on the regular square lattice. Enumeration of polyominoes is an extremely hard problem in enumerative combinatorics, with important applications in statistical physics. We investigate one of the fundamental problems related to polyominoes, namely, computing their asymptotic growth rate $\lambda = \lim_{n\to\infty} \frac{A(n+1)}{A(n)}$, where $A(n)$ is the number of polyominoes of size n. λ is also known as "Klarner's constant." The best lower and upper bounds on λ so far were roughly 3.98 and 4.65, respectively, meaning that not even a single decimal digit of λ was known. Our goal was to settle a long-standing problem: proving that $\lambda > 4$. To this aim we developed a computer program which required extremely high computing resources in terms of both running time and main memory. Our program showed rigorously that $\lambda > 4.00253$.

1 Introduction

λ is a universal constant appearing in the study of three problems studied in seemingly unrelated fields.

Enumerative Combinatorics

Counting polyominoes is a fascinating problem in combinatorics. A *polyomino* of size n is an edge-connected set of n squares on the regular square lattice. Fixed polyominoes are considered distinct if they differ in their shapes or orientations. Figure 1 shows all existing polyominoes, up to translations, rotations, and flips, of size 5. In the mathematical literature, the number of polyominoes of size n is usually denoted by $A(n)$. No formula is known for $A(n)$, and researchers [11, 17] have suggested efficient algorithms for computing $A(n)$ for a given value of n. To-date, the sequence $A(n)$ is known up to $n = 56$ [11].

Percolation Processes

Computing the mean cluster density in percolation processes, in particular those of fluid flow in random media [9], is an important problem in statistical physics. In 1957, Broadbent and Hammersley [5] investigated solute diffusing through solvent, molecules penetrating a porous solid, and similar processes, essentially representing space as a lattice with two distinct types of cells. In the physics literature, fixed polyominoes are usually called "(strongly-embedded) lattice animals."

*Dept. of Computer Science, The Technion, Haifa 32000, Israel. E-mail: {barequet,mshalah}@cs.technion.ac.il

†Institut für Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany. E-mail: rote@inf.fu-berlin.de

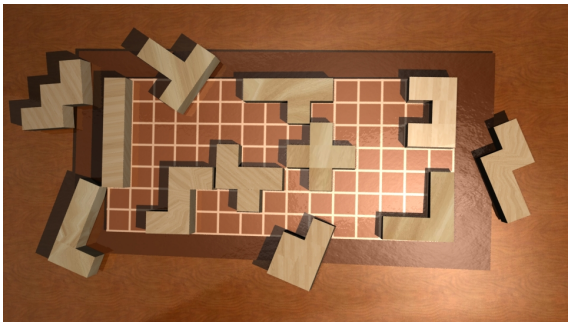

Figure 1: The 12 pentominoes (polyominoes of size 5)

Collapse of Branched Polymers

Another important issue in statistical physics is the existence of a collapse transition of branched polymers in dilute solution at a high temperature [16]. Flesia et al. [8] noted first the relation between the free energy in the process to percolation theory. Derrida and Herrmann [7] investigated two-dimensional branched polymers by looking at lattice animals on the square lattice. They performed exact calculations of the energy of the animal, showing the experimentally known phenomenon of the collapse of branched polymers. Madras et al. [15] also studied branched polymers and their free energy by modeling them as lattice animals.

1.1 The History of λ

Determining the exact value of λ (or even setting good bounds on it) is an extremely hard problem in enumerative combinatorics. In 1967, Klarner [12] showed that the limit $\lambda := \lim_{n\to\infty} \sqrt[n]{A(n)}$ exists, and since then λ has been called "Klarner's constant." Only in 1999 it was proven by Madras [14] that the real asymptotic growth rate, the limit $\lim_{n\to\infty} \frac{A(n+1)}{A(n)}$, exists, and hence it is equal to λ.

By using interpolation methods, Sykes and Glen [18] *estimated* in 1976 that $\lambda = 4.06 \pm 0.02$, an estimate which was improved by Guttmann [10] in 1982 to 4.0626 ± 0.0002, by Conway and Guttmann [6]

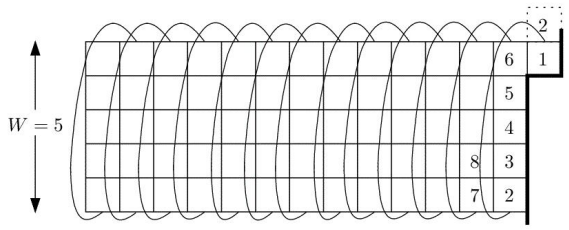

Figure 2: A twisted cylinder of perimeter (width) $W = 5$

in 1995 to 4.06265 ± 0.00005, and by Jensen [11] in 2003 to 4.0625696 ± 0.0000005.

Before carrying out this project, the best proven bounds on λ were roughly 3.9801 [3] from below and 4.6496 [13] from above. Thus, λ has always been an elusive constant, of which not even a single significant digit was (rigorously) known. Our goal was to up the lower bound on λ over the mythical barrier of 4, revealing the identity of its first decimal digit.

2 Twisted Cylinders

A "twisted cylinder" is a half-infinite wrap-around spiral-like square lattice, as is shown in Figure 2.

We denote the perimeter (sometimes called "width" in the literature) of the twisted cylinder by the symbol W. Like in the plane, one can count polyominoes on a twisted cylinder of width W, and ask what their asymptotic growth constant, λ_W, is. It was proven [3] that the sequence $(\lambda_W)_{W=1}^{\infty}$ is monotone increasing, and that $\lambda_W \leq \lambda$ for any value of W. Thus, applying an elementary calculus theorem, this sequence converges. It was later proven [1] that the limit of this sequence is indeed λ. Thus, the bigger W is, the better (higher) the lower bound on λ is. It is easier to analyze the growth rate of polyominoes on a twisted cylinder than to analyze their growth rate in the plane. Polyominoes on a twisted cylinder can be built by considering one square at a time in a uniform way; therefore, the incremental build-up of polyominoes can be modeled very conveniently. Imagine that we walk along the spiral order of squares, and at each square decide whether or not to add it to the polyomino. Naturally, the size of a polyomino is the number of positive decisions we make on the way. The crucial observation is that no matter how big polyominoes are, they can be characterized in a finite number of ways. This is because all we need to remember is the structure of the last W squares of the twisted cylinder (the "boundaries" of the polyominoes), and how they are inter-connected before the boundary. This provides enough information for the continuation of the process: whenever a new square c is considered, and a decision is taken about whether or not to add c to the polyomino, the boundary is updated accordingly. This implies that we can model the growth of polyominoes on a twisted cylinder by a finite-state automaton whose states are all possible boundaries. Every state in this automaton has two outgoing edges that correspond to whether or not the next square is added to the polyomino.

The number of states is extreme [2, 3]: it is equal to M_{W+1}, the $(W+1)$st Motzkin number. M_W increases roughly by a factor of 3 when W is increased by 1. In 2004 we were able to obtain good approximations of λ_W up to $W = 22$. The program that computed λ_W required extremely high computing resources by the standards of that time, both in terms of running time and main memory. For computing $\lambda_{22} \approx 3.9801$, the computation time was about 6 hours on a machine with 32 GB of main memory (RAM). (As of today, the same program runs in 20 minutes on a workstation.) Based on methods from numerical analysis, we extrapolated the first 22 known values of the sequence (λ_W) and estimated already then that only when we reach $W = 27$ we would be able to break the mythical 4.0 barrier. However, with the exponential growth of both memory consumption and running time, this goal seemed at that time to be out of reach.

3 Method

In 2004, Ms. Ares Ribó (a Ph.D. student of G. Rote at that time) developed a serial program that computes λ_W for any perimeter. The program computes and saves the identities of the edges outgoing from any state of the automaton in two long arrays *succ0* and *succ1*, that corresponded to adding an empty or an occupied cell to the polyomino. Both arrays are of length $M = M_{W+1}$. Two arrays of floating point numbers, called y^{old} and y^{new}, store the two successive iteration vectors (that contain the number of polyominoes corresponding to each boundary), also of length M. After initializing $y^{\text{old}} := (1, 1, 1, \dots)$, each iteration computes the new version of y by performing the following simple loop.

$y^{\text{new}}[0] := 0;$
for $s := 1, \dots, M-1$:
$(*)$ $\quad y^{\text{new}}[s] := y^{\text{new}}[\mathit{succ0}[s]] + y^{\text{old}}[\mathit{succ1}[s]];$

The vectors are indexed from 0 to $M-1$. As explained in the previous section, each entry represents a state. The states are encoded by so-called Motzkin paths, and these paths can be bijectively mapped to numbers between 0 and $M-1$.

After each iteration, the contents of y^{new} is moved to y^{old}. The latter vector is normalized after every few iterations in order to prevent overflow of its entries.

Denote by y_{min} (resp., y_{max}) the minimum (resp., maximum) ratio between corresponding entries in y^{new} and y^{old}, that is, $y_{min} := \min_s y^{\text{new}}[s]/y^{\text{old}}[s]$ and $y_{max} := \max_s y^{\text{new}}[s]/y^{\text{old}}[s]$. It was proven [3]

that after every iteration we have $y_{min} \leq \lambda_W \leq y_{max}$, and that in the course of this proceudre y_{min} (resp., y_{max}) is monotone increasing (resp., decreasing), converging (by Perron-Frobenius theorem) to λ_W. Therefore, after every some nominal number of iterations, the program checks whether y_{min} and y_{max} are close enough. In case this test is successful, the program declares convergence.

4 Computing λ_{27}

4.1 Environment

The computation was performed on a Hewlett Packard DL980 G7 computer which consisted of 8 Intel Xeon X7560 nodes (Intel64 architecture), each having eight physical 2.26 GHz processors (16 virtual cores), for a total of 64 processors (128 virtual cores). Each node was equipped with 256 GB of RAM (and 24 MB of cache memory), for a total of 2.048 TB of RAM. It is important to note that simultaneous access by all processors to the shared main memory was crucial to the success of the project. Distributed memory would not work well due to a severe penalty in running time. Hyperthreading was used to allow processes to run on the physical cores of a node while sharing certain resources. Moreover, we had access to a fast disk connected directly to the supercomputer, whose capacity was 3.7 TB (DDR).[1] The Ubuntu Gnu/Linux operating system was run on both computers. Compilation was done using the `gcc` C compiler with OpenMP 3.0 compiler directives for parallel processing.

4.2 Programming Tricks and Improvements

Since for $W = 27$ the finite automaton has roughly $M_{28} = 2.1 \cdot 10^{11}$ states, our initial estimation was that we would need memory for two 8-byte arrays of length M_{28} (for storing *succ0* and *succ1*) and two 4-byte arrays of the same length (for storing y^{old} and y^{new}), for a total of $24 \cdot 2.1 \cdot 10^{11} \approx 5$ TB of RAM, which was certainly out of reach, even with the available supercomputer. We detail below the various ways we improved our program in order to achieve our goal.

4.2.1 Parallelization

The set of states G of the automaton can be partitioned into groups $G_1, G_2 \ldots, G_W$, such that *succ0*$[s]$ for an element $s \in G_i$ belongs to G_{i+1}. (More precisely, the set G_i contains all the states in which the ist cell is the *first* occupied cell along the boundary.) This means that the groups $G_W, \ldots, G_2, G_1$ have to be processed sequentially (in this order), but all elements in one group can be computed in parallel.

We also parallelized the preprocessing phase (computing the *succ* arrays) and various house-keeping tasks.

4.2.2 Elimination of unreachable states

A considerable portion of the states of the automaton (about 11% asymptotically) are unreachable, that is, there is no binary string leading to these states. This happens because not all seemingly legal states can be realized by a valid boundary.

4.2.3 Bit-streaming of the *succ0/1* arrays

Instead of storing each entry of the *succ0/1* arrays in a full word (8 bytes, once the number of states exceeded 2^{32}), we allocated to each entry exactly the number of required bits and stored all entries consecutively in a packed manner. Since the *succ0/1* entries were accessed sequentially, there was only a small overhead in running time for unpacking the resulting bit sequence.

In addition, since we knew a priori to which group G_i each pointer (value in the entry) belonged, we needed only $\lceil \log_2 |G_i| \rceil$ bits per pointer, for all entries in G_i (plus a negligible amount of bits required to delimit between the different sets G_i).

On top of that, the *succ0*-pointer is often illegal because the choice of not adding the next cell to the polyomino caused a connected component of the polyomino to lose contact with the boundary. By spending one extra indicator bit per pointer, we eliminated altogether these illegal pointers, which comprised about 11% of all *succ0* entries.

4.2.4 Storing higher groups only once

If a state s is **not** in the group G_1, then $y^{\text{old}}[s]$ is not needed in the recursion. Thus, we did not need to keep in memory all the entries of the groups G_i, for $i > 1$.

4.2.5 Recomputing *succ0*

Instead of storing the *succ0* array, we computed its entries on-the-fly whenever they were needed, and thus saved completely the array needed to store these pointers. This resulted, naturally, by additional running time for recomputing these pointers whenever they were needed. However, just testing whether *succ0*$[s]$ exists for a given state s could be performed very quickly. Streamlined computation of the pointers accelerated the successor computation (see below). This variation has also benefited from parallelization since each processor can do the pointer computations independently.

[1] This was an important tool since we were allocated "windows" of computation time on the supercomputer, by the end of which we needed to write to disk the intermediate results of the program (i.e., the last contents of y), from which the program could resume in the next running window.

4.2.6 Streamlining the conversion between Motzkin paths and integers

Originally, the Motzkin paths were represented by a sequence of $W+1$ integer numbers taking values from $\{-1, 0, +1\}$. However, we compressed the representation into a sequence of $(W+1)$ 2-bit items, each one encoding one step of the path, which we could store in one 8-byte word (since we had $W \leq 31$). These precomputed data were stored in look-up tables.

This compact storage opened up the possibility of word-level operations. For converting Motzkin paths to numbers, we could process a certain number of steps at a time. Conversely, for converting numbers to Motzkin paths, compressing a certain number of layers (in the path) meant to locate the number in one of at most 3^k intervals of k steps. For each interval we stored the resulting level and the intermediate path. This could be sped up by a table look-up of the leading bits. Using this approach, we obtained a speed-up of 1/3 in the successor computations.

4.3 Execution

On Sunday, May 19, 2013, at 8:25pm Germany time, after performing 120 iterations, the program announced the lower bound 4.00059 on λ_{27}, thus, breaking the 4 barrier. In May 23, after a total of about 36 hours and 290 iterations, the program reached the stable situation (observed in a few successive tens of iterations) $4.002537727 \leq \lambda_{27} \leq 4.002542973$, establishing the new record $\lambda > 4.00253$.

5 Validity and Certification of the Result

The fact that we performed the computations with 32-digit floating-point numbers does **not** imply that numerical errors can be too large so as to nullify our result. Our computed number is an eigenvalue of a huge *integer* matrix. The amount and length of the computations are irrelevant to the fact that eventually we have a witness array of floating-point numbers (the "proof"), about 450 GB in size, which is a good approximation of the eigenvector corresponding to λ_{27}. This array proves rigorous bounds on the true eigenvalue with numerical errors whose magnitude is comparable with the accuracy of floating-point numbers, much smaller than the gap we opened above 4.

We wrote two independent programs for checking this "proof" based on programs for the successor computation written by different people. These programs worked purely sequentially and took about 20 hours each to run. They simply carried out one iteration $(*)$. The result is calculated from the data by at most 26 additions of positive numbers plus one division, all in a single-precision `float`. Since no denormalized numbers occurred, we could estimate the relative error caused by the floating-point operations. We obtained 4.00253176 as a certified lower bound on λ.

Acknowledgment

This project has been carried out on the DL980 G7 supercomputer at HPI (Hasso Plattner Institut) Future SOC Lab in Potsdam, Germany.

References

[1] G. Aleksandrowicz, A. Asinowski, G. Barequet, and R. Barequet, Formulae for polyominoes on twisted cylinders, *LATA*, Madrid, Spain, March 2014.

[2] G. Barequet and M. Moffie, On the complexity of Jensen's algorithm for counting fixed polyominoes, *J. of Discrete Algorithms*, 5 (2007), 348–355.

[3] G. Barequet, M. Moffie, A. Ribó, and G. Rote, Counting polyominoes on twisted cylinders, *INTEGERS: Elec. J. of Comb. Number Theory*, 6 (2006), #A22, 37 pp.

[4] G. Barequet and M. Shalah, Polyominoes on twisted cylinders, *Video Review at the 29th Ann. ACM Symp. on Computational Geometry*, 339–340, June 2013.

[5] S.R. Broadbent and J.M. Hammersley, Percolation processes: I. Crystals and mazes, *Proc. Cambridge Philosophical Society*, 53 (1957), 629–641.

[6] A.R. Conway and A.J. Guttmann, On two-dimensional percolation, *J. Physics, A: Mathematical and General*, 28 (1995), 891–904.

[7] B. Derrida and H.J. Herrmann, Collapse of branched polymers, *J. Physique*, 44 (1983), 1365–1376.

[8] S. Flesia, D.S. Gaunt, C.E. Soteros, and S.G. Whittington, Statistics of collapsing lattice animals, *J. Physics, A: Math. and General*, 27 (1994), 5831–5846.

[9] D.S. Gaunt, M.F. Sykes, and H. Ruskin, Percolation processes in d-dimensions, *J. of Physics A: Mathematical and General*, 9 (1976), 1899–1911.

[10] A.J. Guttmann, On the number of lattice animals embeddable in the square lattice, *J. Physics, A: Mathematical and General*, 15 (1982), 1987–1990.

[11] I. Jensen, Counting polyominoes: A parallel implementation for cluster computing, *Proc. Int. Conf. on Computational Science*, part III, Melbourne, Australia and St. Petersburg, Russia, *Lecture Notes in Computer Science*, 2659, Springer, 203–212, June 2003.

[12] D.A. Klarner, Cell growth problems, *Canadian J. of Mathematics*, 19 (1967), 851–863.

[13] D.A. Klarner and R.L. Rivest, A procedure for improving the upper bound for the number of n-ominoes, *Canadian J. of Mathematics*, 25 (1973), 585–602.

[14] N. Madras, A pattern theorem for lattice clusters, *Annals of Combinatorics*, 3 (1999), 357–384.

[15] N. Madras, C.E. Soteros, S.G. Whittington, J.L. Martin, M.F. Sykes, S. Flesia, and D.S. Gaunt, The free energy of a collapsing branched polymer, *J. Physics, A: Mathematical and General*, 23 (1990), 5327–5350.

[16] P.J. Peard and D.S. Gaunt, $1/d$-expansions for the free energy of lattice animal models of a self-interacting branched polymer, *J. Physics, A: Mathematical and General*, 28 (1995), 6109–6124.

[17] D.H. Redelmeier, Counting polyominoes: Yet another attack, *Discrete Mathematics*, 36 (1981), 191–203.

[18] M.F. Sykes and M. Glen, Percolation processes in two dimensions: I. Low-density series expansions, *J. Physics, A: Mathematical and General*, 9 (1976), 87–95.