

DOI:10.1145/2851485

The universal constant λ , the growth constant of polyominoes (think Tetris pieces), is rigorously proved to be greater than 4.

BY GILL BAREQUET, GÜNTER ROTE, AND MIRA SHALAH

$$\lambda > 4$$

An Improved Lower Bound on the Growth Constant of Polyominoes

What is λ ? The universal constant λ arises in the study of three completely unrelated fields: combinatorics, percolation, and branched polymers. In combinatorics, analysis of self-avoiding walks (SAWs, or non-self-intersecting lattice paths starting at the origin, counted by lattice units), simple polygons or self-avoiding polygons (SAPs, or closed SAWs, counted by either perimeter or area), and “polyominoes” (SAPs possibly with holes, or edge-connected sets of lattice squares, counted by area) are all related. In statistical physics, SAWs and SAPs play a significant role in percolation processes and in the collapse transition that branched polymers undergo when being heated. A collection edited by Guttmann¹⁵ gives an excellent review of all

these topics and the connections between them. In this article, we describe our effort to prove that the growth constant (or asymptotic growth rate, also called the “connective constant”) of polyominoes is strictly greater than 4. To this aim, we exploited to the maximum possible computer resources

» key insights

- Direct access to large shared RAM is useful for scientific computations, as well as for big-data applications.
- The growth constant of polyominoes is provably strictly greater than 4.
- The “proof” of this bound is an eigenvector of a giant matrix Q that can be verified as corresponding to the claimed bound—an eigenvalue of Q .





IMAGE BY RADACHYNSKYI SERHIH

available to us. We eventually obtained a computer-generated proof that was verified by other programs implemented independently. We start with a brief introduction of the three research areas.

Enumerative combinatorics. Imagine a two-dimensional grid of square cells. A polyomino is a connected set of cells, where connectivity is along the sides of cells but not through corners; see Figure 1 for examples of small polyominoes. Polyominoes were popularized in the pioneering book by Golomb¹³ and by Martin Gardner's columns in *Scientific American*; counting polyominoes by size became a popular and fascinating combinatorial problem. The size of a polyomino is the number of its cells.

Figure 2 shows a puzzle game with

polyominoes of sizes 5–8 cells. In this article, we consider “fixed” polyominoes; two such polyominoes are considered identical if one can be obtained from the other by a translation, while rotations and flipping are not allowed. The number of polyominoes of size n is usually denoted as $A(n)$. No formula is known for this number, but researchers have suggested efficient backtracking^{26,27} and transfer-matrix^{9,18} algorithms for computing $A(n)$ for a given value of n . The latter algorithm was adapted in Barequet et al.⁵ and also in this work for polyominoes on so-called twisted cylinders.

To date, the sequence $A(n)$ has been determined up to $n = 56$ by a parallel computation on a Hewlett Packard

server cluster using 64 processors.¹⁸ The exact value of the growth constant of the sequence $\lambda = \lim_{n \rightarrow \infty} A(n+1)/A(n)$ continues to be elusive. Incidentally, in the square lattice, the growth constant is very close to the “coordination number,” or the number of neighbors of each cell in the lattice (4 in our case). In this work we reveal (rigorously) the leading decimal digit of λ . It is 4. Moreover, we establish that λ is strictly greater than the coordination number.

Percolation processes. In physics, chemistry, and materials science, percolation theory deals with the movement and filtering of fluids through porous materials. Giving it a mathematical model, the theory describes the behavior of connected clusters in

Figure 1. The single monomino ($A(1) = 1$), the two dominoes ($A(2) = 2$), the $A(3) = 6$ triominoes, and the $A(4) = 19$ tetrominoes (Tetris pieces).

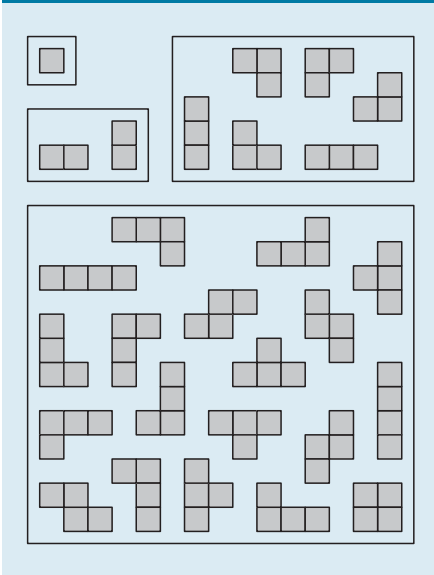
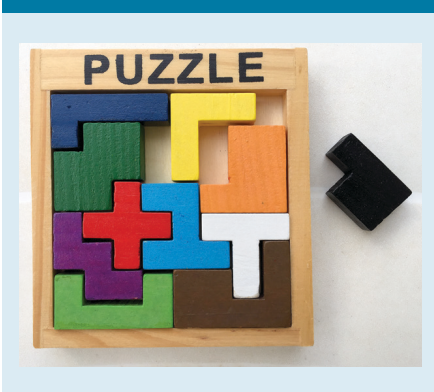


Figure 2. A solitaire game involving six polyominoes of size 5 (pentominoes), two of size 6 (hexominoes), two of size 7 (heptominoes), and one of size 8 (an octomino). The challenge is to tile the 8x8 square box with the polyominoes.



random graphs. Suppose a unit of liquid L is poured on top of some porous material M . What is the chance L makes its way through M and reaches the bottom? An idealized mathematical model of this process is a two- or three-dimensional grid of vertices (“sites”) connected with edges (“bonds”), where each bond is independently open (or closed) for liquid flow with some probability μ . In 1957, Broadbent and Hammersley⁸ asked, for a fixed value of μ and for the size of the grid tending to infinity, what is the probability that a path consisting of open bonds exists from the top to the bottom. They essentially investigated solute diffusing through solvent, molecules penetrat-

ing a porous solid, and similar processes, representing space as a lattice with two distinct types of cells.

In the literature of statistical physics, fixed polyominoes are usually called “strongly embedded lattice animals,” and in that context, the analogue of the growth rate of polyominoes is the growth constant of lattice animals. The terms high and low “temperature” mean high and low “density” of clusters, respectively, and the term “free energy” corresponds to the natural logarithm of the growth constant. Lattice animals were used for computing the mean cluster density in percolation processes, as in Gaunt et al.,¹² particularly in processes involving fluid flow in random media. Sykes and Glen²⁹ were the first to observe that $A(n)$, the total number of connected clusters of size n , grows asymptotically like $C\lambda^n n^\theta$, where λ is Klarner’s constant and C, θ are two other fixed values.

Collapse of branched polymers. Another important topic in statistical physics is the existence of a collapse transition of branched polymers in dilute solution at a high temperature. In physics, a “field” is an entity each of whose points has a value that depends on location and time. Lubensky and Isaacson²² developed a field theory of branched polymers in the dilute limit, using statistics of (bond) lattice animals (important in the theory of percolation) to imply when a solvent is good or bad for a polymer. Derrida and Herrmann¹⁰ investigated two-dimensional branched polymers by looking at lattice animals on a square lattice and studying their free energy. Flesia et al.¹¹ made the connection between collapse processes to percolation theory, relating the growth constant of strongly embedded site animals to the free energy in the processes. Several models of branched polymers in dilute solution were considered by Madras et al.,²⁴ proving bounds on the growth constants for each such model.

Brief History. Determining the exact value of λ (or even setting good bounds on it) is a notably difficult problem in enumerative combinatorics. In 1967, Klarner¹⁹ showed that the limit $\lambda = \lim_{n \rightarrow \infty} A(n)^{1/n}$ exists. Since then, λ has been called “Klarner’s constant.” Only in 1999, Madras²³ proved the stronger statement that the asymptotic growth

rate in the sense of the limit $\lambda = \lim_{n \rightarrow \infty} A(n+1)/A(n)$ exists.

By using interpolation methods, Sykes and Glen²⁹ estimated in 1976 that $\lambda = 4.06 \pm 0.02$. This estimate was refined several times based on heuristic extrapolation from the increasing number of known values of $A(n)$. The most accurate estimate— 4.0625696 ± 0.0000005 —was given by Jensen¹⁸ in 2003. Before carrying out this project, the best proven bounds on λ were roughly 3.9801 from below⁵ and 4.6496 from above.²⁰ λ has thus always been an elusive constant, of which not even a single significant digit was known. Our goal was to raise the lower bound on λ over the barrier of 4, and thus reveal its first decimal digit and prove that $\lambda \neq 4$. The current improvement of the lower bound on λ to 4.0025 also cuts the difference between the known lower bound and the estimated value of λ by approximately 25%—from 0.0825 to 0.0600.

Computer-Assisted Proofs

Our proof relies heavily on computer calculations, thus raising questions about its credibility and reliability. There are two complementary approaches for addressing this issue: formally verified computations and certified computations.

Formally verified computing. In this paradigm, a program is accompanied by a correctness proof that is refined to such a degree of detail and formality it can be checked mechanically by a computer. This approach and, more generally, formally verified “mathematics,” has become feasible for industry-level software, as well as for mathematics far beyond toy problems due to big advances in recent years; see the review by Avigad and Harrison.² One highlight is the formally verified proof by Gonthier¹⁴ of the Four-Color Theorem, whose original proof by Appel and Haken¹ in 1977 already relied on computer assistance and was the prime instance of discussion about the validity and appropriateness of computer methods in mathematical proofs. (Incidentally, one step in Gonthier’s proof also involves polyominoes.) Another example is the verification of Hales’s proof²¹ of the Kepler conjecture, which states the face-centered cubic pack-

ing of spheres is the densest possible. This proof, in addition to involving extensive case enumerations at different levels, is also very complicated in the interaction between the various parts. In August 2014, a team headed by Hales announced the completion of the Flyspeck project, constructing a formal proof of Kepler’s conjecture.¹⁶ Yet another example is the proof by Tucker³⁰ for Lorenz’s conjecture (number 14 in Smale’s list of challenging problems for the 21st century). The conjecture states that Lorenz’s system of three differential equations, providing a model for atmospheric convection, supports a strange attractor. Tucker (page 104)³⁰ described the run of a parallel ODE solver several times on different computer setups, obtaining similar results.

Certified Computation. This technique is based on the idea it may be easier to check a given answer for correctness than come up with such an answer from scratch. The prototype example is the solution of an equation like $3x^3 - 4x^2 - 5x + 2 = 0$. While it may be a challenge to find the solution $x = 2$, it is straightforward to substitute the solution into the equation and check whether it is fulfilled. The result is trustworthy not because it is accompanied by a formal proof but because it is so simple to check, much simpler than the algorithm (perhaps Newton iteration in this case) used to solve the problem in the first place. In addition to the solution itself, it may be required that a certifying algorithm provide a certificate in order to facilitate the checking procedure.²⁵ Developing such certificates and computing them without excessive overhead may require algorithmic innovations (see the first sidebar “Certified Computations”).

In our case, the result computed by our program can be interpreted as the eigenvalue of a giant matrix Q , which is not explicitly stored but implicitly defined through an iteration procedure. The certificate is a vector v that is a good-enough approximation of the corresponding eigenvector. From this vector, one can compute certified bounds on the eigenvalue in a rather straightforward way by comparing the vector v to the product Qv .

Certified Computations

A certifying algorithm not only produces the result but also justifies its correctness by supplying a certificate that makes it easy to check the result. In contrast with formally verified computation, correctness is established for a particular instance of the problem and a concrete result. Here, we illustrate this concept with a few examples; see the survey by McConnell et al.²⁵ for a thorough treatment.

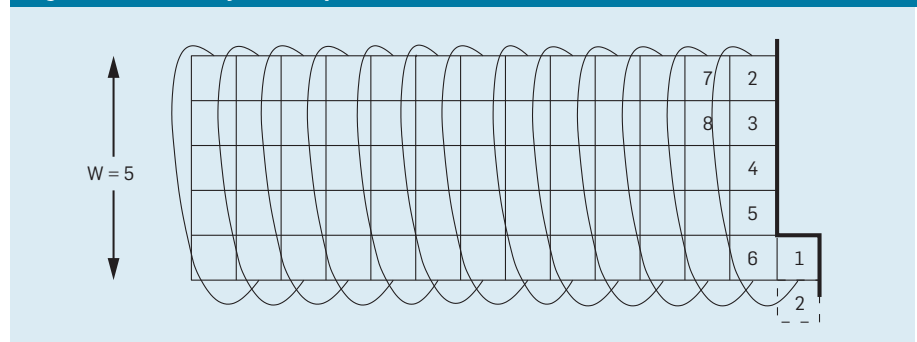
The greatest common divisor. The greatest common divisor of two numbers can be found through the ancient Euclidean algorithm. For example, the greatest common divisor of 880215 and 244035 is 15. Checking that 15 is indeed a common divisor is rather easy, but not clear is that 15 is the greatest. Luckily, the “extended” Euclidean algorithm provides a certificate: two integers $p = -7571$ and $q = 27308$, such that $880215p + 244035q = 15$. This proves any common divisor of 880215 and 244035 must divide 15. No number greater than 15 can thus divide 880215 and 244035.

Systems of linear equations and inequalities. Consider the three equations $4x - 3y + z = 2$, $3x - y + z = 3$, and $x - 7y - z = 4$ in three unknowns x, y, z . It is straightforward to verify any proposed solution; however, the equations have no solution. Multiplying them by 4, -5, -1, respectively, and adding them up leads to the contradiction $0 = -11$. The three multipliers thus provide an easy-to-check certificate for the answer. Such multipliers can always be found for an unsolvable linear system and can be computed as a by-product of the usual solution algorithms. A well-known extension of this example is linear programming, the optimization of a linear objective function subject to linear equations and inequalities, where an optimality certificate is provided by the dual solution.

Testing a graph for 3-connectedness. Certifying that a graph is not 3-connected is straightforward. The certificate consists of two vertices whose removal disconnects the graph into several pieces. It has been known since 1973 that a graph can be tested for 3-connectedness in linear time,¹⁷ but all algorithms for this task are complicated. While providing a certificate in the negative case is easy, defining an easy-to-check certificate in the positive case and finding such a certificate in linear time has required graph-theoretic and algorithmic innovations.²⁸ This example illustrates that certifiability is not primarily an issue of running time. All algorithms, for constructing, as well as for checking, the certificate, run in linear time, just like classical non-certifying algorithms. The crucial distinction is that the short certificate-checking program is by far simpler than the construction algorithm.

Comparison with the class NP. There is some analogy between a certifying algorithm and the complexity class NP. For membership in NP, it is necessary only to have a certificate by which correctness of a solution can be checked in polynomial time. It does not matter how difficult it is to come up with the solution and find the certificate. In contrast with a certifying algorithm, the criterion for the checker is not simplicity but more clear-cut—running time. Another difference is that only “positive” answers to the problem need to be certified.

Figure 3. A twisted cylinder of perimeter $W = 5$.



These two approaches complement each other; a simpler checking procedure is more amenable to a formal proof and verification procedure. However, we did not go to such lengths; a formal verification of the program would have made sense only in the context of a full verification that includes also the human-readable

parts of the proof. We instead used traditional methods of ensuring program correctness of the certificate-checking program.

Twisted Cylinders

A “twisted cylinder” is a half-infinite wraparound spiral-like square lattice (see Figure 3). We denote the perim-

Figure 4. A Motzkin path of length 7.

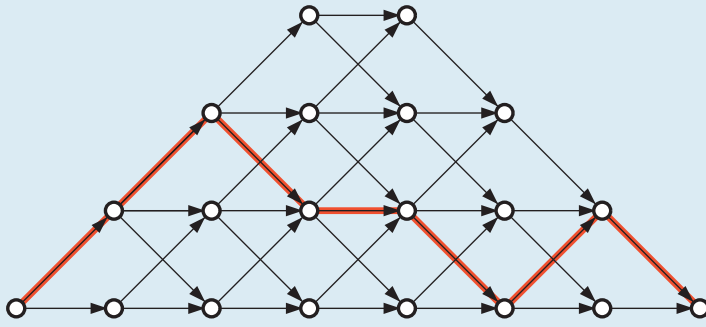
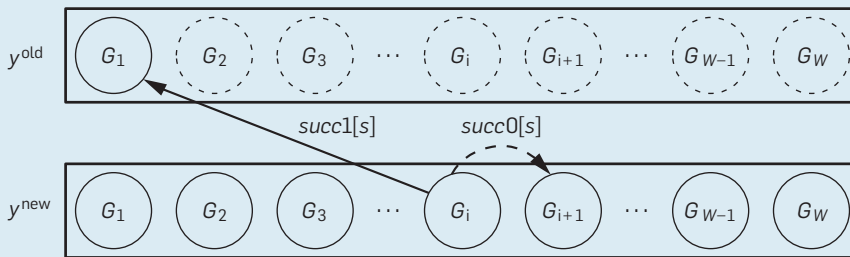


Figure 5. The dependence between the different groups of y^{new} and y^{old} .



eter or “width” of the twisted cylinder by W . Like in the plane, one can count polyominoes on a twisted cylinder of width W and study their growth constant, λ_W . Barequet et al. proved the sequence (λ_W) is monotone increasing⁵ and converges to λ .³ The bigger W is thus the better (higher) the lower bound λ_W on λ gets.

Analyzing the growth constant of polyominoes is more convenient on a twisted cylinder than in the plane. The reason is we want to build up polyominoes incrementally by considering one square at a time. On a twisted cylinder, this can be done in a uniform way, without having to jump to a new row from time to time. Imagine we walk along the spiral order of squares and at each square decide whether or not to add it to the polyomino. The size of a polyomino is the number of positive decisions we make along the way.

The crucial observation is no matter how big the polyominoes get, they can be characterized in a finite number of ways that depends only on W . All one needs to remember is the structure of the last W squares of the twisted cylinder (the “boundary” of the polyomino) and how they are interconnected through cells that were considered

before the boundary. This structure provides enough information for the continuation of the process; whenever a new square is considered and a decision is taken about whether or not to add it to the polyomino, the boundary is updated accordingly. This update is similar to the computation of connected components in a bi-level image by a row-wise scan. In this way, the growth of polyominoes on a twisted cylinder can be modeled by a finite-state automaton whose states are all possible boundaries. Every state of this automaton has two outgoing edges that correspond to whether or not the next square is added to the polyomino. A slight variation of this automaton can be seen in action in a video animation.⁷ See the online appendix for more on automata for modeling polyominoes on twisted cylinders. See also the second sidebar “Representing Boundaries as Motzkin Paths.”

The number of states of the automaton that models the growth of polyominoes on a twisted cylinder of perimeter W is large^{4,5}—the $(W + 1)^{\text{st}}$ Motzkin number M_{W+1} . The n^{th} Motzkin number M_n counts Motzkin paths of length n ; see Figure 4 for an illustra-

tion of a Motzkin path and the sidebar for the relation between states and Motzkin paths. Such a path connects the integer grid points $(0,0)$ and $(n,0)$ with n steps, consisting only of steps taken from $\{(1,1), (1,0), (1,-1)\}$ and not going under the x axis. Asymptotically, $M_n \sim 3^n n^{-3/2}$, and M_W thus increases roughly by a factor of 3 when W is incremented by 1.

The number of polyominoes with n cells that have state s as the boundary equals the number of paths the automaton can take from the starting state to s , involving n transitions in which a cell is added to the polyomino. We compute these numbers in a dynamic-programming recursion.

Method

In 2004, a sequential program that computes λ_W for any perimeter was developed by Ares Ribó as part of her Ph.D. thesis under the supervision of author Günter Rote. The program first computes the endpoints of the outgoing edges from all states of the automaton and saves them in two long arrays *succ0* and *succ1* that correspond to adding an empty or an occupied cell. Both arrays are of length $M := M_{W+1}$. Two successive iteration vectors, which contain the number of polyominoes corresponding to each boundary, are stored as two arrays y^{old} and y^{new} of numbers, also of length M . The four arrays are indexed from 0 to $M - 1$. After initializing $y^{\text{old}} := (1,0,0,\dots)$, each iteration computes the new version of y through a very simple loop:

```

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

```

The pointer *succ0*[s] may be null, in which case the corresponding zero entry ($y^{\text{new}}[0]$) is used.

As explained earlier, each index s represents a state. The states are encoded by Motzkin paths, and these paths can be mapped to numbers between 0 and $M - 1$ in a bijective manner. See the online appendix for more.

In the iteration (*), the vector y^{new} depends on itself, but this does not

cause any problem because $succ0[s]$, if it is non-null, is always less than s . There are thus no circular references, and each entry is set before it is used. In fact, the states can be partitioned into groups G_1, G_2, \dots, G_W ; the group G_i contains the states corresponding to boundaries in which i is the smallest index of an occupied cell, or the boundaries that start with $i - 1$ empty cells. The dependence between the entries of the groups is shown schematically in Figure 5; $succ0[s]$ of an entry $s \in G_i$ (for $1 \leq i \leq W - 1$), if it is non-null, belongs to G_{i+1} .

At the end, y^{new} is moved to y^{old} to start the new iteration. In the iteration (*), the new vector y^{new} is a linear function of the old vector y^{old} and, as already indicated, can be written as a linear transformation $y^{old} := Qy^{old}$. The nonnegative integer matrix Q is implicitly given through the iteration (*). We are interested in the growth rate of the sequence of vectors y^{old} that is determined by the dominant eigenvalue λ_w of Q . It is not difficult to show⁵ that after every iteration, λ_w is contained in the interval

$$\min_s \frac{y^{new}[s]}{y^{old}[s]} \leq \lambda_w \leq \max_s \frac{y^{new}[s]}{y^{old}[s]} . \quad (1)$$

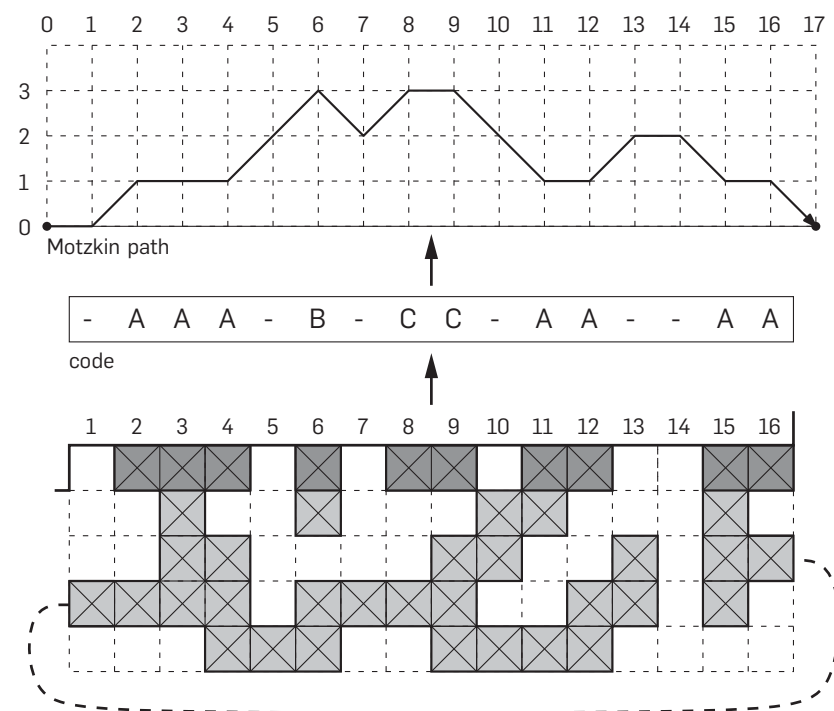
By the Perron-Frobenius Theorem about the eigenvalues of nonnegative matrices, the two bounds converge to λ_w , and y^{old} converges to a corresponding eigenvector.

As written, the program calculates the exact number of polyominoes of each size and state, provided the computations are carried out with precise integer arithmetic. However, these numbers grow like $(\lambda_w)^n$, and the program can thus not afford to store them exactly. Instead, we use single-precision floating-point numbers for y^{old} and y^{new} . Even so, the program must rescale the vector y^{old} from time to time in order to prevent floating-point overflow: Whenever the largest entry exceeds 2^{80} at the end of an iteration, the program divides the whole vector by 2^{100} . This rescaling does not affect the convergence of the process. The program terminates when the two bounds are close enough. The left-hand side of (1) is a lower bound on λ_w , which in turn is a lower bound on λ , and this is our real goal.

Representing Boundaries as Motzkin Paths

The figure here illustrates the representation of polyomino boundaries as Motzkin paths. The bottom part of the figure shows a partially constructed polyomino on a twisted cylinder of width 16. The dashed line indicates two adjacent cells that are connected “around the cylinder,” where such a connection is not immediately apparent. The boundary cells (top row) are shown in darker gray. The light-gray cells away from the boundary need not be recorded individually; what matters is the connectivity among the boundary cells they provide. This connectivity is indicated in a symbolic code -AAA-B-CC-AA--AA. Boundary cells in the same component are represented by the same letter, and the character ‘-’ denotes an empty cell. However, we did not use this code in our program. Instead, we represented a boundary as a Motzkin path, as shown in the top part of the figure, because this representation allows for a convenient bijection to successive integers and thus for a compact storage of the boundary in a vector. Intuitively, the Motzkin path follows the movements of a stack when reading the code from left to right. Whenever a new component starts (such as component A in position 2 or component B in position 6), the path moves up. Whenever a component is temporarily interrupted (such as component A in position 5), the path also moves up. The path moves down when an interrupted component is resumed (such as component A in positions 11 and 15) or when a component is completed (positions 7, 10, and 17). The crucial property is that components cannot cross; that is, a pattern like . . A . B . A . B . cannot occur. As a consequence of these rules, the occupied cells correspond to odd levels in the path, and the free cells correspond to even levels. The correspondence between boundaries and Motzkin paths was pointed out by Stefan Felsner of the Technische Universität Berlin (private communication).

Polyomino boundaries and Motzkin paths.



Sequential Runs

In 2004, we obtained good approximations of y^{old} up to $W = 22$. The program required quite a bit of main memory (RAM) by the standards of the time. The computation of $\lambda_{22} \approx$

3.9801 took approximately six hours on a single-processor machine with 32GB of RAM. (Today, the same program runs in 20 minutes on a regular workstation.) By extrapolating the first 22 values of the sequence λ_w

(see Figure 6), we estimated that only when we reach $W = 27$ would we break the mythical barrier of 4.0. However, as mentioned earlier, the storage requirement is proportional to M_w , and M_w increases roughly by a factor of 3 when W is incremented by 1. With this exponential growth of both memory consumption and running time, the goal of breaking the barrier was then out of reach.

Computing λ_{27}

Environment. In the spring of 2013, we were granted access to the Hewlett Packard ProLiant DL980 G7 server of the Hasso Plattner Institute Future SOC Lab in Potsdam, Germany (see Figure 7). This server consists of eight Intel Xeon X7560 nodes (Intel64 architecture), each with eight physical 2.26GHz processors (16 virtual cores), for a total of 64 processors (128 virtual

cores). Each node was equipped with 256GiB of RAM (and 24MiB of cache memory) for a total of 2TiB of RAM. Simultaneous access by all processors to the shared main memory was crucial to project success. Distributed memory would incur a severe penalty in running time. The machine ran under the Ubuntu version of the Gnu/Linux operating system. We used the C-compiler gcc with OpenMP 2.0 directives for parallel processing.

Programming improvements. Since for $W = 27$ the finite automaton has $M_{28} \approx 2.1 \cdot 10^{11}$ states, we initially estimated we would need memory for two 8-byte arrays (for storing *succ0* and *succ1*) and two 4-byte arrays (for storing y^{old} and y^{new}), all of length M_{28} , for a total of $24 \cdot 2.1 \cdot 10^{11} \approx 4.6$ TiB of RAM, which exceeded the server’s available capacity. Only a combination of parallelization, storage-compression techniques,

and a few other enhancements allowed us to compute λ_{27} and push the lower bound on λ above 4.0. Here are the main memory-reduction tricks we used (see the online appendix for more):

Elimination of unreachable states. Approximately 11% of all states of the automaton are unreachable and do not affect the result;

Bit-streaming of the succ0/1 arrays. Instead of storing each entry of these arrays in a full word, we allocated just the required number of bits and stored the entries consecutively in a packed manner; in addition, we eliminated all the null *succ0*-pointers (approximately 11% of all pointers);

Storing higher groups only once. Approximately 50% of the states (those not in G_1) were not needed in the recursion (*); we thus did not keep in memory $y^{old}[\cdot]$ of the respective entries;

Recomputing succ0. We computed the entries of the *succ0* array on-the-fly instead of keeping them in memory; and

Streamlining the successor computation. Instead of representing a Motzkin path by a sequence of $W + 1$ integers, we kept a sequence of $W + 1$ two-bit items we could store in one 8-byte word; this representation allowed us to use word-level operations and look-up tables for a fast mapping from paths to numbers.

To make full use of the parallel capacities, we used the partition of the set of states into groups, as in Figure 5, such that y^{new} of all elements in a group could be computed independently. We also parallelized the computation of the *succ* arrays in the preprocessing phase and various housekeeping tasks.

Execution. After 120 iterations, the program announced the lower bound 4.00064 on λ_{27} , thus breaking the 4 barrier. We continued to run the program for a few more days. On May 23, 2013, after 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 $\lambda > 4.00253$. The total running time for the computations leading to this result was approximately 36 hours using approximately 1.5TiB of RAM. We had exclusive use of the server for a few dozen hours in total, spread over several weeks.

Figure 6. Extrapolating the sequence λ_w .

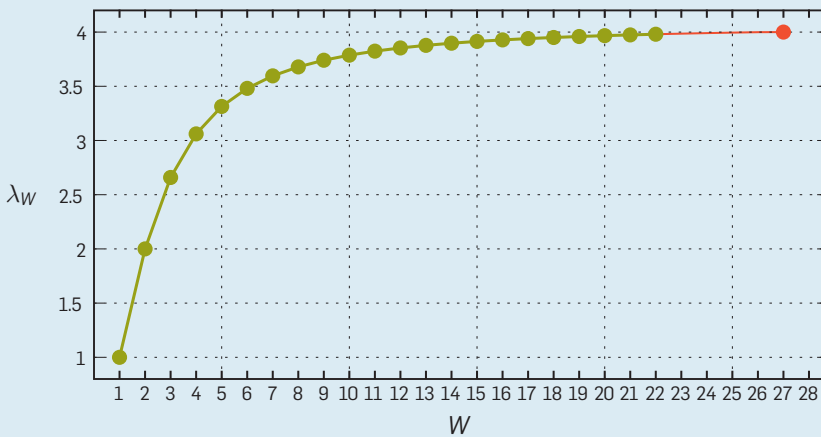
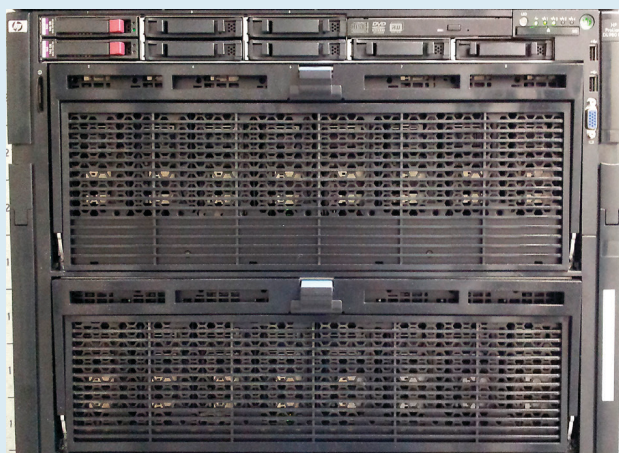


Figure 7. Front view of the “supercomputer” we used, a box of approximately 45×35×70 cm.



Validity and Certification

Our proof depends heavily on computer calculations, raising two issues about its validity:

Calculations. Reproducing elaborate calculations on a large computer is difficult; particularly when a complicated parallel computer program is involved, everybody should be skeptical about the reliability of the results; and

Computations. We performed the computations with 32-bit floating-point numbers.

We address these issues in turn.


What our program was trying to compute is an eigenvalue of a matrix. The amount and length of the computations are irrelevant to the fact that eventually we have stored on disk a witness array of floating-point numbers (the “proof”), approximately 450GB in size, which is a good approximation of the eigenvector corresponding to λ_{27} . This array provides rigorous bounds on the true eigenvalue λ_{27} , because the relation (1) holds for *any* vector y^{old} and its successor vector y^{new} . To check the proof and evaluate the bounds (1), a program has to read only the approximate eigenvector y^{old} and carry out one iteration (*). This approach of providing simple certificates for the result of complicated computations is the philosophy of “certifying algorithms.”²⁵

To ensure the correctness of our checking program, we relied on traditional methods (such as testing and code inspection). Some parts of the program (such as reading the data from the files) are irrelevant for the correctness of the result. The main body of the program consists of a few simple loops (such as the iteration (*) and the evaluation of (1)). The only technically challenging part of the algorithm is the successor computation. For this task, we had two programs at our disposal that were written independently by two people who used different state representations—and lived in different countries and did their work several years apart. We ran two different checking programs based on these procedures, giving us additional confidence. We also tested explicitly that the two successor programs yielded the same results. Both checking programs ran in a purely sequential manner, and the

running time was approximately 20 hours each.

Regarding the accuracy of the calculations, one can analyze how the recurrence (*) produces y^{new} from y^{old} . One finds that each term in the lower bound (1) results from the input data (the approximate eigenvector y^{old}) through at most 26 additions of positive numbers for computing $y^{\text{new}}[s]$, plus one division, all in single-precision float. The final minimization was error-free. Since we took care that no denormalized floating-point numbers occurred, the magnitude of the numerical errors was comparable to the accuracy of floating-point numbers, and the accumulated error was thus much smaller than the gap we opened between our new bound and 4. By carefully bounding the floating-point error, we obtained 4.00253176 as a certified lower bound on λ . In particular, we thus now know that the leading digit of λ is 4.

Acknowledgments

We acknowledge the support of the facilities and staff of the Hasso Plattner Institute Future SOC Lab in Potsdam, Germany, who let us use their Hewlett Packard ProLiant DL980 G7 server. A technical version of this article⁶ was presented at the 23rd Annual European Symposium on Algorithms in Patras, Greece, September 2015. 

References

- Appel, K. and Haken, W. Every planar map is four colorable. *Illinois Journal of Mathematics* 21, 3 (Sept. 1977), 429–490 (part I) and 491–567 (part II).
- Avigad, J. and Harrison, J. Formally verified mathematics. *Commun. ACM* 57, 4 (April 2014), 66–75.
- Aleksandrowicz, G., Asinowski, A., Barequet, G., and Barequet, R. Formulae for polyominoes on twisted cylinders. In *Proceedings of the Eighth International Conference on Language and Automata Theory and Applications, Lecture Notes in Computer Science 8370* (Madrid, Spain, Mar. 10–14). Springer-Verlag, Heidelberg, New York, Dordrecht, London, 2014, 76–87.
- Barequet, G. and Moffie, M. On the complexity of Jensen’s algorithm for counting fixed polyominoes. *Journal of Discrete Algorithms* 5, 2 (June 2007), 348–355.
- Barequet, G., Moffie, M., Ribó, A., and Rote, G. Counting polyominoes on twisted cylinders. *INTEGERS: Electronic Journal of Combinatorial Number Theory* 6 (Sept. 2006), #A22, 37.
- Barequet, G., Rote, G., and Shalah, M. $\lambda > 4$. In *Proceedings of the 23rd Annual European Symposium on Algorithms Lecture Notes in Computer Science 9294* (Patras, Greece, Sept. 14–16). Springer-Verlag, Berlin Heidelberg, Germany, 2015, 83–94.
- Barequet, G. and Shalah, M. Polyominoes on twisted cylinders. In the *Video Review at the 29th Annual Symposium on Computational Geometry* (Rio de Janeiro, Brazil, June 17–20). ACM Press, New York, 2013, 339–340; <http://www.computational-geometry.org/SoCG-videos/socg13video>
- Broadbent, S.R. and Hammersley, J.M. Percolation processes: I. Crystals and mazes. *Proceedings of the Cambridge Philosophical Society* 53, 3 (July 1957), 629–641.

- Conway, A. Enumerating 2D percolation series by the finite-lattice method: Theory. *Journal of Physics, A: Mathematical and General* 28, 2 (Jan. 1995), 335–349.
- Derrida, B. and Herrmann, H.J. Collapse of branched polymers. *Journal de Physique* 44, 12 (Dec. 1983), 1365–1376.
- Flesia, S., Gaunt, D.S., Soteris, C.E., and Whittington, S.G. Statistics of collapsing lattice animals. *Journal of Physics, A: Mathematical and General* 27, 17 (Sept. 1994), 5831–5846.
- Gaunt, D.S., Sykes, M.F., and Ruskin, H. Percolation processes in d-dimensions. *Journal of Physics A: Mathematical and General* 9, 11 (Nov. 1976), 1899–1911.
- Golomb, S.W. *Polyominoes, Second Edition*. Princeton University Press, Princeton, NJ, 1994.
- Gonthier, G. Formal proof—The four color theorem. *Notices of the AMS* 55, 11 (Dec. 2008), 1382–1393.
- Guttman, A.J., Ed. *Polygons, Polyominoes and Polycubes, Lecture Notes in Physics* 775. Springer, Heidelberg, Germany, 2009.
- Hales, T., Solovveyev, A., and Le Truong, H. *The Flyspeck Project: Announcement of Completion*, Aug. 10, 2014; <https://code.google.com/p/flyspeck/wiki/AnnouncingCompletion>
- Hopcroft, J.E. and Tarjan, R.E. Dividing a graph into triconnected components. *SIAM Journal of Computing* 2, 3 (Sept. 1973), 135–158.
- Jensen, I. Counting polyominoes: A parallel implementation for cluster computing. In *Proceedings of the International Conference on Computational Science, Part III, Lecture Notes in Computer Science, 2659* (Melbourne, Australia, and St. Petersburg, Russian Federation, June 2–4). Springer-Verlag, Berlin, Heidelberg, New York, 2003, 203–212.
- Klarner, D.A. Cell growth problems. *Canadian Journal of Mathematics* 19, 4 (1967), 851–863.
- Klarner, D.A. and Rivest, R.L. A procedure for improving the upper bound for the number of *n*-ominoes. *Canadian Journal of Mathematics* 25, 3 (Jan. 1973), 585–602.
- Lagarias, J.C., Ed. *The Kepler Conjecture: The Hales-Ferguson Proof*. Springer, New York, 2011.
- Lubensky, T.C. and Isaacson, J. Statistics of lattice animals and dilute branched polymers. *Physical Review A* 20, 5 (Nov. 1979), 2130–2146.
- Madras, N. A pattern theorem for lattice clusters. *Annals of Combinatorics* 3, 2–4 (June 1999), 357–384.
- Madras, N., Soteris, C.E., Whittington, S.G., Martin, J.L., Sykes, M.F., Flesia, S., and Gaunt, D.S. The free energy of a collapsing branched polymer. *Journal of Physics, A: Mathematical and General* 23, 22 (Nov. 1990), 5327–5350.
- McConnell, R.M., Mehlhorn, K., Näher, S., and Schweitzer, P. Certifying algorithms. *Computer Science Review* 5, 2 (May 2011), 119–161.
- Mertens, S. and Lautenbacher, M.E. Counting lattice animals: A parallel attack. *Journal of Statistical Physics* 66, 1–2 (Jan. 1992), 669–678.
- Redelmeier, D.H. Counting polyominoes: Yet another attack. *Discrete Mathematics* 36, 3 (Dec. 1981), 191–203.
- Schmidt, J.M. Contractions, removals and certifying 3-connectivity in linear time. *SIAM Journal on Computing* 42, 2 (Mar. 2013), 494–535.
- Sykes, M.F. and Glen, M. Percolation processes in two dimensions: I. Low-density series expansions. *Journal of Physics, A: Mathematical and General* 9, 1 (Jan. 1976), 87–95.
- Tucker, W. A rigorous ODE solver and Smale’s 14th problem. *Foundations of Computational Mathematics* 2, 1 (Jan. 2002), 53–117.

Gill Barequet (barequet@cs.technion.ac.il) is an associate professor in and vice dean of the Department of Computer Science at the Technion - Israel Institute of Technology, Haifa, Israel.

Günter Rote (rote@inf.fu-berlin.de) is a professor in the Department of Computer Science at Freie Universität Berlin, Germany.

Mira Shalah (mshalah@cs.technion.ac.il) is a Ph.D. student, under the supervision of Gill Barequet, in the Department of Computer Science at the Technion - Israel Institute of Technology, Haifa, Israel.