Available online at www.sciencedirect.com
ScienceDirect
Journal of Discrete Algorithms 5 (2007) 348–355
JOURNAL OF DISCRETE ALGORITHMS
www.elsevier.com/locate/jda

On the complexity of Jensen's algorithm for counting fixed polyominoes ☆

Gill Barequet [a,*,1], Micha Moffie [b]

[a] *Department of Computer Science, The Technion—Israel Institute of Technology, Haifa 32000, Israel*
[b] *Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115, USA*

Received 6 November 2004; accepted 2 August 2005

Available online 21 July 2006

Abstract

Recently I. Jensen published a novel transfer-matrix algorithm for computing the number of polyominoes in a rectangular lattice. However, his estimation of the computational complexity of the algorithm ($O((\sqrt{2})^n)$, where n is the size of the polyominoes), was based only on empirical evidence. In contrast, our research provides some solid proof. Our result is based primarily on an analysis of the number of some class of strings that plays a significant role in the algorithm. It turns out that this number is closely related to Motzkin numbers. We provide a rigorous computation that roughly confirms Jensen's estimation. We obtain the bound $O(n^{5/2}(\sqrt{3})^n)$ on the running time of the algorithm, while the actual number of polyominoes is about $C4.06^n/n$, for some constant $C > 0$.

Keywords: Lattice animals; Polyominoes; Computational complexity

1. Introduction

A polyomino of size n is an edge-connected set of n squares on a regular square lattice. *Fixed* polyominoes are considered distinct if they have different shapes *or* orientations. The symbol $A(n)$ usually denotes in the literature the number of fixed polyominoes of size n. The first few values of this series are $\{A(n)\}_{n=1}^{\infty} = \{1, 2, 6, 19, 63, 216, 760, \ldots\}$. Polyominoes have triggered the imagination of many scientists, not only mathematicians. The issue of the number of fixed polyominoes arises, for instance, when investigating the properties of liquid flow through grained material [3], such as water flowing through coffee grains. Statistical physicists refer to polyominoes as *lattice animals*, whose number is relevant when computing the mean cluster density in percolation processes. To this day there is no known analytic formula for $A(n)$. The only known methods for computing $A(n)$ are based on explicitly or implicitly enumerating all the polyominoes.

☆ A preliminary version of this paper appeared in [G. Barequet, M. Moffie, The complexity of Jensen's algorithm for counting polyominoes, in: Proc. 1st Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, January 2004, pp. 161–169. [1]].

* Corresponding author.
E-mail addresses: barequet@cs.technion.ac.il (G. Barequet), mmoffie@ece.neu.edu (M. Moffie).

[1] Work on this paper by the first author has been supported in part by the Fund for the Promotion of Research at the Technion.

1570-8667/$ – see front matter
doi:10.1016/j.jda.2005.08.001

The following is a brief overview of the developments of counting fixed polyominoes:

- In 1962 Read [21] derived generating functions for calculating the number of fixed polyominoes. These functions become intractable very fast and were used for computing $A(n)$ for only $n = 1, \ldots, 10$ (with an error in $A(10)$).
- In 1967 Parkin et al. [19] computed the number of polyominoes of up to size 15 (with a slight error in $A(15)$) on a CDC 6600 computer.
- In 1971 Lunnon [14] computed the values of $A(n)$ up to $n = 18$ (with a slight error in $A(17)$). His program generated polyominoes that could fit into a restricted rectangle. Since his algorithm generated the same polyominoes more than once, the program spent a considerable amount of time checking polyominoes for repetitions. It was thus an $O(nA^2(n))$-time algorithm, and the program ran for about 175 hours (a little more than a week) on a Chilton Atlas I.
- An algorithm of Martin [15] and Redner [23] that computes polyominoes of a given size and perimeter was used in 1976 by Sykes and Glen [24] to enumerate $A(n)$ up to $n = 19$.
- In 1981 Redelmeier [22] introduced a novel enumeration algorithm. His new method, which was actually a procedure for subgraph counting, did not reproduce any of the previously-generated polyominoes. Thus, he did not need to keep in memory all the already-generated polyominoes, nor did he need to check if a newly-generated polyomino was already counted. He implemented his method efficiently, and computed $A(n)$ up to $n = 24$. It was the first $O(A(n))$-time algorithm, and Redelmeier's program required about 10 months of CPU time on a PDP-11/70. Mertens and Lautenbacher [16] later devised a parallel version of Redelmeier's algorithm and used it for computing the number of *equal-perimeter* polyominoes (of up to perimeter 22) on a triangular lattice.
- In 1995 Conway [4] introduced a transfer-matrix algorithm, subsequently used by Conway and Guttmann [5] for computing $A(25)$.
- Oliveira e Silva [18] used (in an unpublished work) a parallel version of Redelmeier's algorithm to count *free* polyominoes (for which the orientation does not matter) of up to size 28.
- Jensen [7] significantly improved the algorithm of Conway and Guttmann, and computed $A(n)$ up to $n = 46$. Following quickly, Knuth [12] applied (in an unpublished work) a few local optimizations to Jensen's algorithm and was able to compute $A(47)$. In a further computation Jensen [8] obtained $A(48)$. Finally, Jensen [9] also parallelized the algorithm and computed $A(n)$ for $n \leqslant 56$.

It is known that $A(n)$ is exponential in n. Klarner [10] showed that the limit $\lambda = \lim_{n\to\infty} \sqrt[n]{A(n)}$ exists. It is widely believed, being supported by the 56 currently-known values of $A(n)$, that $A(n) \sim C\lambda^n n^\theta$ for some constants $C > 0$ and $\theta \approx -1$ (see [13]). So, the limit $\lim_{n\to\infty}(A(n+1)/A(n))$ also exists and is equal to λ. The number λ is thus called the "growth rate" of polyominoes, and was given its well-known name Klarner's constant by Golomb [6]. At the moment, not even a single significant digit of λ is known for sure! There have been several attempts to lower and upper bound λ, as well as to estimate it, based on knowing the first values of $A(n)$. The best-known published lower and upper bounds are 3.927378 [9][2] and 4.649551 [11]. Recently, a new lower bound of 3.980137 was shown in [2]. The constant λ is estimated to be around 4.06 ± 0.02 [5,9].

In this paper we analyze the running time of Jensen's Algorithm [7]. It has two factors: a polynomial in n, and an exponential term $J(W)$ (where $W = n/2$). Jensen did not provide an analysis of the running time of his algorithm. Instead, he showed a graph [ibid., Section 2.1.3] that plots $J(W)$ (the number of signatures of length W, a key ingredient of the algorithm—see Section 3.1) as a function of W, for $1 \leqslant W \leqslant 23$. The graph was drawn in a half-logarithmic scale, and the plotted points visually seem to be more-or-less located around some line. From the slope of this imaginary line, Jensen estimated that $J(W)$ was proportional to 2^W. Therefore, Jensen concluded that the running time of his algorithm was proportional to $(\sqrt{2})^n$ times some polynomial factor. Jensen's algorithm *prunes* the signatures, discarding intermediate configurations that cannot be completed into valid polyominoes. In this paper we analyze $S(W)$, the number of unpruned signatures. We show that it equals $M(W+1) - 1$, where $M(W)$ is the Wth Motzkin number [17]. Since the Wth Motzkin number is proportional to 3^W (neglecting some polynomial factor), we obtain an upper bound on the running time of Jensen's algorithm, which is proportional to $(\sqrt{3})^n$ times some polynomial factor.

[2] It is pointed out in [2] that this lower bound is based on an incorrect assumption, which goes back to a paper by Rands and Welsh [20]. The correct lower bound should have been 3.87565.

This paper is organized as follows. In Section 2 we give a brief description of Jensen's transfer-matrix algorithm for counting fixed polyominoes, and in Section 3 we provide a detailed analysis of the running time of the algorithm.

2. Jensen's transfer-matrix algorithm

In this section we briefly describe Jensen's algorithm for counting fixed polyominoes. The reader is referred to the original paper [7] for the full details.

The algorithm separately counts all the polyominoes of size n bounded by rectangles whose dimensions are W (its width, y-span) and L (its length, x-span). In each iteration of the algorithm different values of W and L are considered, for all possible values of W and L. (Due to symmetry, only values of $W \leqslant L$ need to be considered.)

For specific values of W and L, the strategy is as follows. The polyominoes are built from left to right, and in each column from top to bottom. Instead of keeping track of all polyominoes, the procedure keeps records of the numbers of polyominoes with identical right boundaries. Towards this aim the right boundaries of the (yet incomplete) polyominoes are encoded by signatures as will be described shortly. A polyomino is expanded in the current column, cell by cell, from top to bottom. The new cell is either occupied (i.e., belongs to the new polyomino) or empty (i.e., does not belong to it). Thus, the right boundaries of the (yet incomplete) polyominoes have a "kink" at the currently considered cell. By "expanding" we mean updating both the signatures (possibly creating new signatures) and their respective numbers of polyominoes. For implementation purpose the numbers are maintained as polynomials in the form of generating functions: The terms of the polynomial $P(t) = \sum_i c_i t^i$ mean that c_i distinct (possibly incomplete) polyominoes of size i correspond to some signature.

The right boundaries of (yet incomplete) polyominoes are encoded by signatures that contain five symbols: the digits 0–4. The symbol '0' stands for an empty cell. The symbol '1' stands for an occupied cell that is not connected by other cells to any other boundary cells. The other three symbols represent cells which are connected to other boundary cells. In case there are several boundary cells that are connected either along the boundary or by cells to the left of it, the lowest cell is encoded by the symbol '2', the highest cell by the symbol '4', and all the other cells (if any) in that group by the symbol '3'. Fig. 1 (similar to Fig. 1 of [7]) shows the signature of some (yet incomplete) polyomino, before and after cell expansion. Note that this polyomino is indeed incomplete because it has three disconnected components. Yet the algorithm needs to consider such intermediate configurations since the yet-to-be-added cells on the right may connect disconnected components and yield a legal polyomino. Only at the termination of the iteration does the algorithm need to check whether a signature corresponds to legal polyominoes. Otherwise, the corresponding number of polyominoes (of that signature) is ignored and not counted. Also note in the figure that the lowest '2' and the highest '4' in the signature encode boundary cells which are connected by cells to the left of the boundary.

There are several implementation details that are omitted here. One such detail is the initialization of an iteration, that is, building the set of signatures that correspond to (yet incomplete) polyominoes spanning only one column. Another detail is the exhaustive set of polyomino-expansion rules. In an expansion step the boundary kink is lowered by one by considering the kink cell and either making it occupied or leaving it empty. The symbols encoding the cells adjacent to the kink, together with the fact of whether or not the new cell is occupied, determine how to update the signature. Most combinations require local updates, whereas a few combinations require scanning the entire signature for updating. In addition, the signature needs to be expanded by two bits that indicate whether or not (yet incomplete) polyominoes touch the top and bottom boundaries of the bounding rectangle.

Some important optimizations are also performed. Since only polyominoes of size n are sought, the signatures of intermediate polyominoes exceeding this size may be discarded. In addition, local rules can be applied to prune out intermediate configurations that cannot be completed into valid polyominoes. For example, it would save time to discard signatures of intermediate polyominoes that have more than one connected component, and whose union into one component will require a total of more than n cells. The same applies for signatures of intermediate polyominoes whose yet-to-be-constructed connections to all of the top, bottom, and right borders will require more than n cells in total.

At the termination of each iteration, the procedure discards all signatures that correspond to illegal polyominoes, such as disconnected polyominoes (as mentioned above), or polyominoes that do not touch all the boundaries of the bounding rectangle. Then, all the numbers of polyominoes associated with legal polyominoes are summed up to

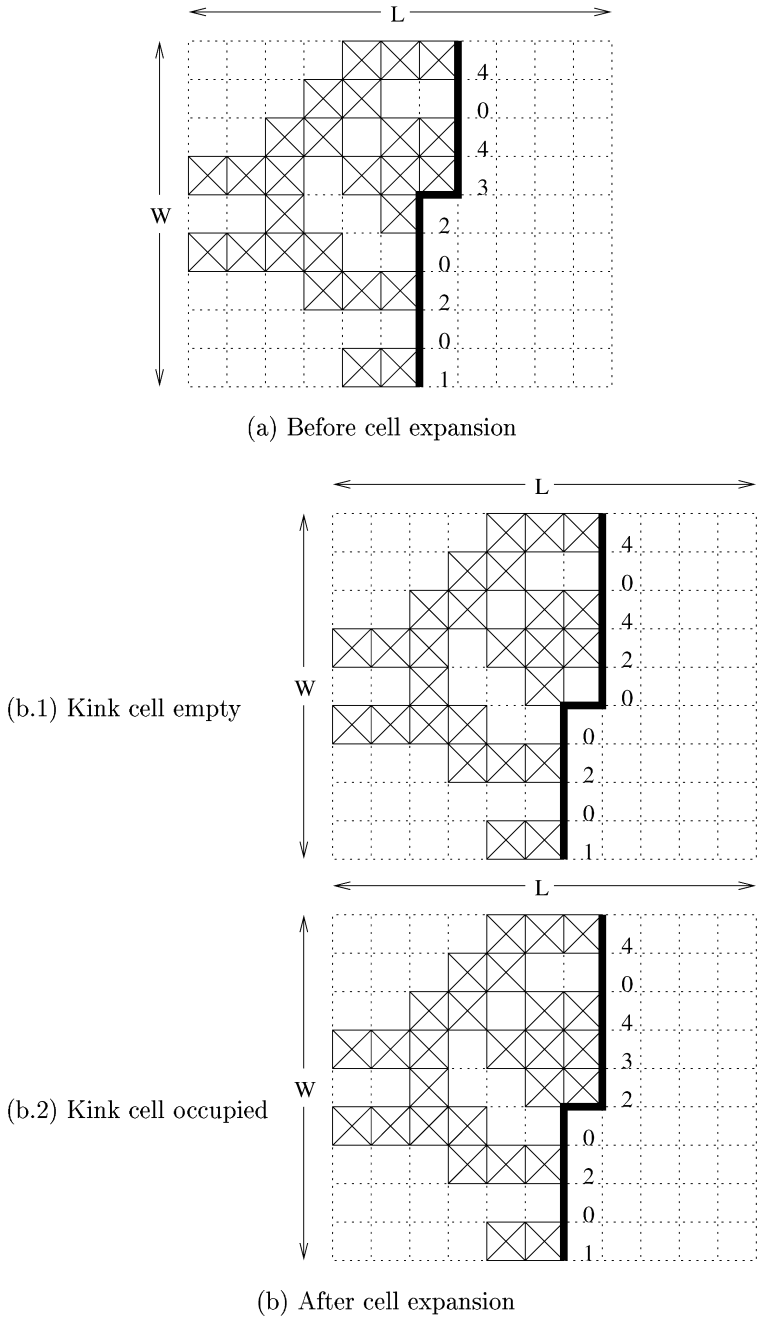

Fig. 1. A sample intermediate polyomino, its right boundary, and its signature (example taken from [7]).

yield the number of legal polyominoes bounded by some $W \times L$ rectangle. By iterating over all possible values of W and L and summing up, the algorithm computes the total number of polyominoes of size n. (Naturally, for one specific value of W, the iterations of the inner loop on L may transfer information between them to save execution time.)

As mentioned in the introduction, Jensen did not provide an analysis of the running time of his algorithm. Instead, he empirically estimated that it is proportional to $(\sqrt{2})^n$, where n is the size of the sought after polyominoes. In the following section we provide a more rigorous estimation of the running time.

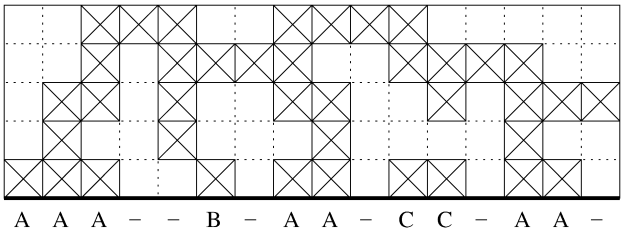

Fig. 2. A lattice configuration and its signature.

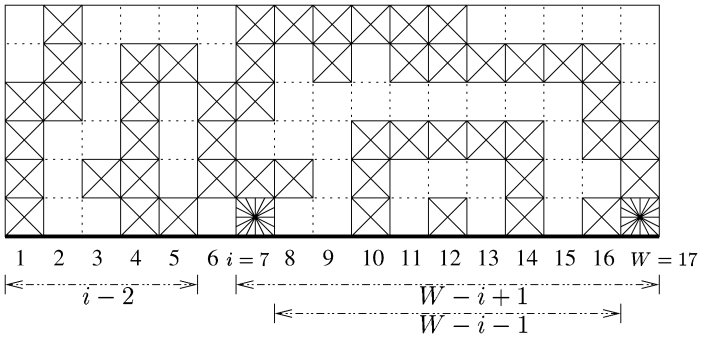

Fig. 3. The leftmost signature cell connected to the Wth signature cell (when the latter is occupied).

3. Computational complexity

3.1. Boundary strings

We begin by investigating the number of kinkless boundary strings of length W. (In this section the boundaries are assumed to be horizontal for ease of exposition.) Consider a square lattice of width W and *unrestricted* height h, in which some of the $W \cdot h$ cells are occupied and the other cells are empty. Identify the connected components of the occupied cells, where connectivity is through edges only. Assign a symbol to each connected component and a special symbol '-' to empty cells, and regard the lowest row of the lattice as the "signature" string representing the configuration of occupied cells. The completely empty string (containing only '-'s) is not allowed. Obviously, a signature may represent many different configurations. Moreover, the number of distinct symbols needed to specify all signatures is $\lceil W/2 \rceil + 1$, since there may be at most $\lceil W/2 \rceil$ connected components. Fig. 2 shows a lattice configuration and its representing signature string. The question is, then, what is $S(W)$, the number of different signature strings (as a function of W, up to renaming of the non-'-' symbols).

It is shown in [2] that $S(W) = M(W+1) - 1$. This is proven by a bijection between signature strings of length W and Motzkin paths of length $W+1$ (except the straight path). We provide here a direct proof of this fact.

Theorem 1. $S(W) = M(W+1) - 1$.

Proof. Let $S^*(W) = S(W) + 1$, that is, the number of all legal signatures plus the unique empty signature (-,-,-, . . . ,-) that is illegal in the context of polyominoes. We will now evaluate $S^*(W)$. The number of signature strings whose last character is '-' is simply $S^*(W-1)$. Otherwise (when the last character is not '-'), the corresponding last cell of the signature may be connected (through the boundary and/or the area to the left of the boundary, which in this section is the area on top of the signature) to other signature cells. Let i be the lowest index (counting from left to right) of the signature cell connected to the Wth cell (obviously $1 \leqslant i \leqslant W$). For $i \geqslant 2$, the $(i-1)$st cell must be empty, and the number of distinct signatures is the product of the number of subsignatures of the leftmost $i-2$ cells and the number of subsignatures of the rightmost $W-i+1$ cells (of which the leftmost and rightmost are occupied by definition, so there is freedom only in the middle $W-i-1$ cells). Fig. 3 shows an example in which $W = 17$ and $i = 7$. (Recall that cell connectivity is through edges only!)

Thus,

$$S^*(W) = S^*(W-1) + \sum_{i=1}^{W} \bigl(S^*(i-2) \cdot S^*(W-i-1)\bigr),$$

with the convention $S^*(-1) = S^*(0) = 1$. It is easily seen that this is exactly the recurrence of the Motzkin series (with a shift of one index):

$$M(k) = M(k-1) + \sum_{i=0}^{k-2} \bigl(M(i) \cdot M(k-i-2)\bigr)$$

for $k \geqslant 2$ and $M(0) = M(1) = 1$. It follows immediately that $S(W) = S^*(W) - 1 = M(W+1) - 1$. Indeed, $\{S(W)\}_{W=1}^{\infty} = \{1, 3, 8, 20, 50, 126, \ldots\}$ while the first few Motzkin numbers are $\{M(k)\}_{k=0}^{\infty} = \{1, 1, 2, 4, 9, 21, 51, 127, \ldots\}$. □

3.2. Running time

We now analyze the running time of Jensen's algorithm, following closely the notation of [7]. Denote by n the size of the polyominoes whose number we intend to compute. Overall, we have p iterations of Jensen's algorithm that differ in the dimensions of the bounding rectangle of the polyominoes counted in each iteration. Denote by W (resp., L) the width (resp., length), that is, the y-span (resp., x-span) of a bounding rectangle in one iteration of Jensen's algorithm. In each such iteration W is at most $W_{\max}$ (the maximum width needed to be considered), and $L = 2W_{\max} + 1 - W$; thus, $L_{\max}$ (the maximum length) is at most $2W_{\max}$. In each iteration of the algorithm we expand the so-far traversed lattice by $W \cdot L$ cells. Each such expansion consists of determining whether a new cell is either occupied or empty. This involves the consideration of all possible signatures σ. Denote the total number of possible signatures by N_{Conf}. Each individual cell-expansion step requires the following operations:

- Fetching a signature from the database of signatures;
- Computing a new signature that is a function of the old signature, and of whether the new cell is occupied or not;
- Updating the polynomial that counts the number of possible distinct configurations for the specific signature; and
- Searching for the new signature in the database. If it did not previously exist, insert it and its polynomial. Otherwise, replace the existing polynomial by the sum of the existing and new polynomial.

We denote by h the time needed for a random access into the signatures database, by s the time for computing a new signature, and by u the time needed for updating a polynomial. Clearly, the over-all running time of the algorithm is

$$\mathrm{O}\bigl(p \cdot W_{\max} \cdot L_{\max} \cdot N_{\mathrm{Conf}} \cdot (h+s+u)\bigr).$$

Obviously, $W_{\max} = \lceil n/2 \rceil$ and $L_{\max} = n$. Therefore, $W_{\max}, L_{\max} = \mathrm{O}(n)$. In the original description of the algorithm, $p = W_{\max} \cdot L_{\max} = \mathrm{O}(n^2)$. However, instead of two nested loops for W and L, one can have only one loop on W with only $L = L_{\max}$. After processing each column, one can check all the configurations. The polynomials representing the respective numbers of polyominoes of all the valid configurations (those with one connected component spanning the entire $W \times L$ rectangle) will update the number of polyominoes of the appropriate sizes. Thus we can improve p by a factor of n and have $p = \mathrm{O}(n)$.

Assume that one implements the database of signatures as a simple (albeit large) sequential RAM array that contains the records of all possible signatures. In order to access a signature, one first needs to convert the signature to its running number (address in the database); this can be done in $\mathrm{O}(W_{\max})$ time [2]. Accessing a signature in the database can be done in constant time (or more precisely, proportionally to the length of the key, which is $\mathrm{O}(W_{\max})$). Even if one needs to perform a binary search in the database, this will require time that is logarithmic in the size of the database. This is $\log N_{\mathrm{Conf}}$, which is, as is shown below, again, $\mathrm{O}(W_{\max})$. Modifying a signature structure takes $\mathrm{O}(W_{\max})$ time too. In conclusion, $h = \mathrm{O}(W_{\max}) = \mathrm{O}(n)$.

It follows directly from the description of the algorithm that $s = \mathrm{O}(W_{\max}) = \mathrm{O}(n)$. Indeed, most of the signature updates are local, and are reflected by $\mathrm{O}(1)$ operations in the vicinity of the expanded cell. Only a few update rules

require the traversal of the entire signature (whose complexity is $O(n)$) and a few global updates of it. A typical such update is required when the expanded cell touches the top boundary cell of some connected component, and we either need to find the second-to-top, or even the bottom boundary cell of the same component. For this purpose we traverse the signature top to bottom and count top and bottom cells of connected components until they balance appropriately, providing enough information for identifying the sought after boundary cell. The running time of this traversal is linear in n.

The polynomial operations required by Jensen's algorithm are the addition of two polynomials $P_1(t)$, $P_2(t)$ of maximum degree n and multiplication of a polynomial $P(t)$ of maximum degree n by t. (The latter operation implements the addition of one occupied cell.) Clearly each such operation requires $O(n)$ time, and since each cell-expansion operation requires a constant number of such polynomial operations, we have $u = O(n)$.

Finally, we provide an upper bound on N_{Conf}, the number of signatures of length W. We ignore the two bits of the signature that indicate whether or not the respective set of polyominoes touch the top and bottom boundaries of the bounding box, because these bits add only a constant factor of 4 on the number of different signatures.

In Section 3.1 we showed that the number of signatures without a kink is exactly $M(W+1)-1$, where $M(k) = \Theta(3^k/k^{3/2})$ is the kth Motzkin number. Now consider boundaries with a kink. At most, this applies a constant factor of 3 on the number of signatures. To see this, regard signatures of length W as a subset of all the signatures of length $(W+1)$ obtained by adding the kink cell to the boundary as an empty cell, and then dropping from the signature the symbol '0' at the position of the kink. By dropping these kink symbols different signatures may be identified, but this only helps. Thus, we conclude that $N_{\text{Conf}}(W) = O(3^W/W^{3/2}) = O((\sqrt{3})^n/n^{3/2})$.

Summing everything up, we obtain $O(n^4 M(n/2)) = O(n^{5/2}(\sqrt{3})^n)$ as an asymptotic bound on the running time of Jensen's algorithm. As mentioned earlier, Jensen's algorithm also prunes out signatures of boundaries that can never close into a legal polyomino because, for example, the number of cells remaining to be occupied is insufficient for the polyomino to touch both upper and lower boundaries of the bounding box, or because there are not enough cells to connect all occupied cells into one connected component. Obviously, most of the signatures are pruned out when W and L are close to $n/2$. This is probably the reason for the difference between the $(\sqrt{3})^n$ term in our bound and the $(\sqrt{2})^n$-like behavior observed empirically by Jensen.

Acknowledgements

We are grateful to Günter Rote and Stefan Felsner for suggesting the use of Motzkin numbers, and to Noga Alon and Dan Romik for providing important insights on these numbers.

References

[1] G. Barequet, M. Moffie, The complexity of Jensen's algorithm for counting polyominoes, in: Proc. 1st Workshop on Analytic Algorithmics and Combinatorics, New Orleans, LA, January 2004, pp. 161–169.
[2] G. Barequet, M. Moffie, A. Ribó, G. Rote, Counting polyominoes on twisted cylinders, in: Proc. 3rd European Conf. on Combinatorics, Graph Theory, and Applications, Berlin, Germany, September 2005, pp. 369–374. Full version: Integers, accepted for publication.
[3] S.R. Broadbent, J.M. Hammersley, Percolation processes: I. Crystals and mazes, Proceedings of the Cambridge Philosophical Society 53 (1957) 629–641.
[4] A.R. Conway, Enumerating 2D percolation series by the finite-lattice method: Theory, Journal of Physics, A: Mathematical and General 28 (1995) 335–349.
[5] A.R. Conway, A.J. Guttmann, On two-dimensional percolation, Journal of Physics, A: Mathematical and General 28 (1995) 891–904.
[6] S.W. Golomb, Polyominoes, second ed., Princeton Univ. Press, Princeton, NJ, 1994.
[7] I. Jensen, Enumerations of lattice animals and trees, Journal of Statistical Physics 102 (2001) 865–881.
[8] I. Jensen, http://sunburn.stanford.edu/~knuth/programs/jensen.txt, a personal communication with D.E. Knuth.
[9] I. Jensen, Counting polyominoes: A parallel implementation for cluster computing, in: Proc. Int. Conf. on Computational Science, Melbourne, Australia, in: Lecture Notes in Computer Science, vol. 2659, Springer-Verlag, Heidelberg, August 2003, pp. 203–212.
[10] D.A. Klarner, Cell growth problems, Canadian Journal of Mathematics 19 (1967) 851–863.
[11] D.A. Klarner, R.L. Rivest, A procedure for improving the upper bound for the number of n-ominoes, Canadian Journal of Mathematics 25 (1973) 585–602.
[12] D.E. Knuth, http://sunburn.stanford.edu/~knuth/programs.html#polyominoes, a personal WWW page.
[13] T.C. Lubensky, J. Isaacson, Statistics of lattice animals and dilute branched polymers, Physical Review A 20 (1979) 2130–2146.
[14] W.F. Lunnon, Counting polyominoes, in: A.O.L. Atkin, B.J. Birch (Eds.), Computers in Number Theory, Academic Press, London, 1971, pp. 347–372.

[15] J.L. Martin, Computer techniques for evaluating lattice constants, in: C. Domb, M.S. Green (Eds.), Phase Transitions and Critical Phenomena, vol. 3, Academic Press, London, 1974, pp. 97–112.
[16] S. Mertens, M.E. Lautenbacher, Counting lattice animals: A parallel attack, Journal of Statistical Physics 66 (1992) 669–678.
[17] T. Motzkin, Relations between hypersurface cross ratios, and a combinatorial formula for partitions of a polygon, for permanent preponderance, and for non-associative products, Bulletin of the American Mathematical Society 54 (1948) 352–360.
[18] T. Oliveira e Silva, http://www.ieeta.pt/~tos/animals/a44.html, a personal WWW page.
[19] T.R. Parkin, L.J. Lander, D.R. Parkin, Polyomino enumeration results, in: SIAM Fall Meeting, Santa Barbara, CA, 1967.
[20] B.M.I. Rands, D.J.A. Welsh, Animals, trees and renewal sequences, IMA Journal of Applied Mathematics 27 (1981) 1–17; Corrigendum: IMA Journal of Applied Mathematics 28 (1982) 107.
[21] R.C. Read, Contributions to the cell growth problem, Canadian Journal of Mathematics 14 (1962) 1–20.
[22] D.H. Redelmeier, Counting polyominoes: Yet another attack, Discrete Mathematics 36 (1981) 191–203.
[23] S. Redner, A Fortran program for cluster enumeration, Journal of Statistical Physics 29 (1982) 309–315.
[24] M.F. Sykes, M. Glen, Percolation processes in two dimensions: I. Low-density series expansions, Journal of Physics, A: Mathematical and General 9 (1976) 87–95.