A parallel algorithm for the enumeration of self-avoiding polygons on the square lattice

2003 J. Phys. A: Math. Gen. 36 5731

(http://iopscience.iop.org/0305-4470/36/21/304)

INSTITUTE OF PHYSICS PUBLISHING JOURNAL OF PHYSICS A: MATHEMATICAL AND GENERAL

J. Phys. A: Math. Gen. **36** (2003) 5731–5745 PII: S0305-4470(03)58841-2

A parallel algorithm for the enumeration of self-avoiding polygons on the square lattice

Iwan Jensen

Department of Mathematics and Statistics, The University of Melbourne, Vic. 3010, Australia

Received 24 January 2003, in final form 21 March 2003
Published 13 May 2003
Online at stacks.iop.org/JPhysA/36/5731

Abstract

We have developed a parallel algorithm that allows us to enumerate the number of self-avoiding polygons on the square lattice to perimeter length 110. We have also extended the series for the first 10 area-weighted moments and the radius of gyration to 100. Analysis of the resulting series yields very accurate estimates of the connective constant $\mu = 2.638\,158\,530\,31(3)$ (biased) and the critical exponent $\alpha = 0.500\,000\,1(2)$ (unbiased). In addition, we obtain very accurate estimates for the leading amplitudes confirming to a high degree of accuracy various predictions for universal amplitude combinations.

PACS numbers: 05.50.+q, 02.10.Hh, 05.40.Fb

1. Introduction

A self-avoiding polygon (SAP) on a lattice can be defined as a walk along the edges of the lattice which starts and ends at the origin but has no other self-intersections. Alternatively, we can define a SAP as a connected subgraph with vertices of degree 2 only. Generally, SAPs are considered distinct up to a translation, so if there are p_n SAPs of length n there are $2np_n$ walks (the factor of two arising since the walk can go in two directions). The enumeration of self-avoiding polygons on various lattices is an interesting combinatorial problem in its own right, and is also of considerable importance in the statistical mechanics of lattice models [15]. When enumerated by perimeter SAPs can be considered a model for ring polymers, and when enumerated by area they model vesicles [10, 11, 21].

The basic problems are the calculation of the number p_n of polygons of perimeter n, the number a_m of polygons of area m, or more generally the number $p_{m,n}$ of polygons of area m and perimeter n. Note that on the square lattice polygons have an *even* perimeter and $p_n = 0$ for n odd. Here we are interested in area-weighted moments, where the kth area-weighted moment is $\langle a^k \rangle_n = \left(\sum_m m^k p_{m,n}\right) \big/ p_n$. Also of great interest is the mean-square radius of

The series for the generating functions studied in this paper can be obtained via e-mail by sending a request to I.Jensen@ms.unimelb.edu.au or via the world wide web on the URL http://www.ms.unimelb.edu.au/~iwan/ by following the instructions.

gyration $\langle R^2\rangle_n$, which measures the typical size of a polygon with perimeter n. These quantities are expected to behave as

$$\begin{aligned} p_n &= B\mu^n n^{\alpha-3}[1+o(1)] \\ \langle a^k\rangle_n &= E^{(k)} n^{2k\nu}[1+o(1)] \\ \langle R^2\rangle_n &= Dn^{2\nu}[1+o(1)] \end{aligned} \tag{1}$$

where μ is the so-called connective constant, while α and ν are critical exponents. When analysing the data, it is often convenient to use the associated generating functions

$$\mathcal{R}_g^2(u) = \sum_n n^2 p_n \langle R^2\rangle_n u^n = \sum_n r_n u^n \sim R(u)(1-u\mu)^{-(\alpha+2\nu)} \tag{2}$$

$$\mathcal{P}^{(k)}(u) = \sum_n p_n \langle a^k\rangle_n u^n = \sum_n a_n^{(k)} u^n \sim A^{(k)}(u)(1-u\mu)^{2-(\alpha+2k\nu)} \tag{3}$$

where the various factors are chosen so that r_n and $a_n^{(k)}$ are integers. These series are thus expected to have a singularity at the critical point $u_c = 1/\mu$ with critical exponents as mentioned above. In particular, we note that the critical exponent of the perimeter generating function, $\mathcal{P}(u) = \mathcal{P}^{(0)}(u)$, is $2-\alpha$.

Despite strenuous effort over the past 50 years or so, this problem has not been solved on any regular two-dimensional lattice. However, for the hexagonal lattice the critical point $u_c^2 = 1/(2+\sqrt{2})$, as well as the critical exponents $\alpha = 1/2$ and $\nu = 3/4$ are known exactly [25], though non-rigorously. Very firm evidence exists from previous numerical work that the exponent α is universal and thus equals $1/2$ for all two-dimensional lattices [9, 13, 17, 24]. The value of ν and its universality have also been confirmed by numerical work [8, 13, 16, 24, 26].

It is also known [2] that the amplitude combination $E^{(1)}/D$ is universal, and that

$$BD = \frac{5}{32\pi^2}\sigma a_0 \tag{4}$$

where a_0 is the area per site, and σ is an integer such that p_n is non-zero only if n is divisible by σ. For the square lattice $a_0 = 1$ and $\sigma = 2$. These predictions have been confirmed numerically [2, 16, 22–24].

Recently, Richard *et al* [27] found, subject to a very reasonable conjecture, the exact scaling function for self-avoiding polygons. This in turn led to the derivation of universal amplitude combinations for *all* the $E^{(k)}$, namely that $E^{(k)}B^{k-1}$ are known universal constants. In particular, it has been shown that $E^{(1)} = 1/4\pi$ [1]. These predictions were strongly supported by numerical evidence [27].

Some years ago [3] it was pointed out that since the hexagonal lattice connective constant is given by the zero of a quadratic in u^2, it is possible that this might be the case also for the square lattice connective constant. It was found that $581u^4 + 7u^2 - 13$ was the only polynomial with 'small' integer coefficients consistent with the estimate for u_c. The relevant zero of the polynomial is $u_c^2 = 0.143\,680\,629\,269\,8685\ldots$. In [18] the numerical evidence was in complete agreement with this value, but with four more significant digits than when the original suggestion was made. While it should be emphasized that there is no theoretical basis for the conjecture, the agreement with the numerical estimate is, however, sufficiently impressive (and perhaps surprising) to warrant further investigation.

This paper builds on the work of Enting [6] who enumerated square lattice polygons to 38 steps using the finite lattice method. Using the same technique this enumeration was extended by Enting and Guttmann to 46 steps [7] and later to 56 steps [13], and further extended to

70 steps in unpublished work. These extensions to the enumeration were largely made possible by improved computer technology. Jensen and Guttmann [18] improved the algorithm and extended the enumeration to 90 steps while using essentially the same computational resources used to obtain polygons to 70 steps. The work by Guttmann and Enting [13] also included calculations of moments of the caliper size distribution. Hiley and Sykes [14] obtained the number of square lattice polygons by both area and perimeter up to perimeter 18. Enting and Guttmann extended the calculation to perimeter 42 [8]. The radius of gyration was calculated for SAPs up to 28 steps by Privman and Rudnick [26]. Jensen [16] extended the series for area-moments with $k \leqslant 2$ and the radius of gyration to 82 steps. In [27] the calculation for area-moments was extended to $k \leqslant 10$.

The main purpose of this paper is to report on a new parallel version of our earlier algorithms which allows us to significantly extend the series and use these extended series to critically examine the theoretical predictions given above. Using the parallelized algorithm and a new superior memory management, inspired by Knuth's work on the enumeration of polyominoes [20], we have been able to extend the enumeration of square lattice polygons to 110 steps. We extend the series for area-weighted moments with $k \leqslant 10$ and the radius of gyration to 100 steps.

In the next section we will very briefly review the finite lattice method for enumerating square lattice polygons and give some details of the improved parallel algorithm. The results of the analysis of the series are presented in section 3 including a discussion of the conjecture for the exact critical point and numerical tests of the predictions for universal amplitude combinations.

2. Enumeration of polygons

The algorithm used to enumerate SAPs on the square lattice is an enhancement of the finite-lattice method devised by Enting [6] in his pioneering work, which contains a detailed description of the original approach. A major enhancement, resulting in an exponentially more efficient algorithm, is described in some detail in [18], while details of the changes required to enumerate area-moments and the radius of gyration can be found in [16]. In the following we shall briefly outline those parts of the method required to understand how the parallel version works.

2.1. The transfer matrix algorithm

The first terms in the series for the polygon generating function can be calculated using transfer matrix techniques to count the number of polygons in rectangles W vertices wide and L vertices long. Due to the symmetry of the square lattice one need only consider rectangles with $L \geqslant W$. In the original application [6] valid polygons were required to span the enclosing rectangle in the lengthwise direction. Clearly, polygons which are narrower than the width of the rectangle are counted many times. It is, however, easy to obtain the polygons of width exactly W and length exactly L from this enumeration [6]. Any polygon spanning such a rectangle has a perimeter of length at least $2(W + L) - 4$. By adding the contributions from all rectangles of width $W \leqslant W_{\max}$ (where the choice of $W_{\max}$ depends on available computational resources) and length $W \leqslant L \leqslant 2W_{\max} - W + 1$, with contributions from rectangles with $L > W$ counted twice, the number of polygons per vertex of an infinite lattice is obtained correctly up to perimeter $N_{\max} = 4W_{\max} - 2$.

The transfer matrix technique involves drawing a boundary line through the rectangle intersecting a set of up to $W + 1$ edges. Polygons in a given rectangle are enumerated by

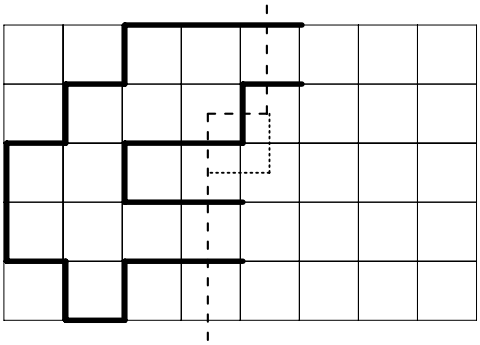

Figure 1. A snapshot of the boundary line (dashed line) during the transfer matrix calculation on the square lattice. Polygons are enumerated by successive moves of the kink in the boundary line, as exemplified by the position given by the dotted line, so that one vertex at a time is added to the rectangle. To the left of the boundary line we have drawn an example of a partially completed polygon.

moving the boundary line so as to add one vertex at a time, as shown in figure 1. In this fashion we build up the rectangle column-by-column with each column built up vertex-by-vertex. As we move the boundary line, it intersects partially completed polygons consisting of disjoint loops that must all be connected to form a single polygon. For each configuration of the occupied or empty edges along the intersection, we maintain a (perimeter) generating function for open loops to the left of the line cutting the intersection in that particular pattern. The updating of the generating functions depends primarily on the configuration of the two edges at the kink in the boundary line prior to the move (we shall refer to these edges as the kink edges). As the boundary line is moved, the two new edges intersected by the boundary line can be either empty or occupied.

To avoid situations leading to graphs with more than a single component, we have to forbid a loop to close on itself if the boundary line intersects any other loops. So two loop ends can only be joined if they belong to different loops or all other edges are empty. To exclude loops which close on themselves, we need to label the occupied edges in such a way that we can easily determine whether or not the two loop ends belong to the same loop. The most obvious choice would be to give each loop a unique label. However, on two-dimensional lattices there is a more compact scheme relying on the fact that two loops can never intertwine. Each end of a loop is assigned one of two labels depending on whether it is the lower end or the upper end of a loop. Each configuration along the boundary line can thus be represented by a set of edge states $\{\sigma_i\}$, where

$$\sigma_i = \begin{cases} 0 & \text{empty edge} \\ 1 & \text{lower end of a loop} \\ 2 & \text{upper end of a loop.} \end{cases} \tag{5}$$

Configurations are read from the bottom to the top. The configuration along the intersection of the partially completed polygon in figure 1 is $\{011\overline{2}\overline{1}22\}$ before the move, where we use over-lining to indicate the kink edges, and $\{01\overline{1}\overline{0}022\}$ after the move. It is easy to see that this encoding uniquely describes which loop ends are connected. In order to find the upper loop end, matching a given lower end, we start at the lower end and work upwards in the configuration counting the number of '1's and '2's we pass (the '1' of the initial lower end is *not* included in the count). We stop when the number of '2's exceeds the number of '1's. This '2' marks the matching upper end of the loop. Ignoring the '0's, the '1's and '2's can

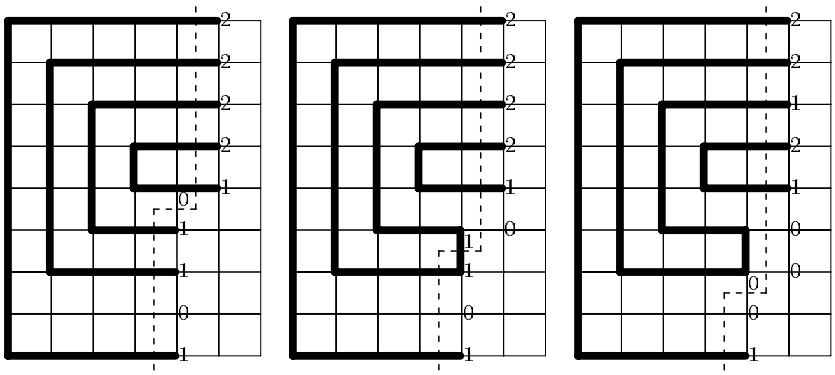

Figure 2. Snapshots of the boundary line (dashed line) during the TM calculation. Shown is a situation with four nested loops (left panel) where the lower ends of two loops are joined (middle panel) resulting in a situation with three nested loops (right panel) and a relabelling of the loop ends.

be viewed as perfectly balanced parenthesis. Those familiar with algebraic languages will recognize that each configuration of the labelled loop ends forms a Motzkin word [5]. It is known that the number of Motzkin words of length m grows like 3^m. This means that the number of configurations and thus the computational complexity of the FLM calculation grows like $3^{N_{\max}/4}$.

The rules for updating the partial generating functions as the intersection is moved are identical to the original work, so we refer the interested reader to [6] for further details regarding this aspect of the transfer-matrix calculation. The only important aspect we wish to emphasize here is that when joining two loop ends at the kink, we may have to change the labelling of a corresponding loop end elsewhere in the resulting new configuration. An example is shown in figure 2. In this case we start out with four nested loops corresponding to the configuration $\{101\overline{1}012222\}$, then upon moving the kink in the boundary line the lower loop ends of the second and third loops are joined leading to the configuration $\{10\overline{11}012222\}$. After the next move we see that there are now three differently nested loops and the upper end of the second loop (prior to the moves) have become the lower end of the third loop (after the moves) resulting in the final configuration $\{1\overline{00}0012122\}$.

The major improvement to the original method as explained in [18] is that we require valid polygons to span the rectangle in *both* directions. In other words, we directly enumerate polygons of width exactly W and length L rather than polygons of width $\leqslant W$ and length L as was done originally. At a first glance this would appear to be inefficient since for many boundary-line configurations, we now have to keep four distinct generating functions depending on which borders have been touched. However, as demonstrated in practice [18] it actually leads to an algorithm which is both exponentially faster and whose memory requirement is exponentially smaller. Experimentally it was found that the computational complexity was close to $2^{N_{\max}/4}$, much better than the $3^{N_{\max}/4}$ of the original approach. Realizing the full savings in time and memory usage requires enhancements to the original algorithm. The most important is what we call *pruning*. This procedure, details of which are given in [18], allows us to discard most of the possible configurations for large W because they only contribute to polygons of length greater than $N_{\max}$. Briefly this works as follows. Firstly, for each configuration we keep track of the current minimum number of steps N_{cur}, already inserted to the left of the boundary line to build up that particular configuration. Secondly, we calculate the minimum number of additional steps N_{add} required

to produce a valid polygon. There are three contributions, namely the number of steps required to close the polygon, the number of steps needed (if any) to ensure that the polygon touches both the lower and upper borders and finally the number of steps needed (if any) to extend at least W edges in the lengthwise direction (remember we only need rectangles with $L \geqslant W$). If the sum $N_{\mathrm{cur}} + N_{\mathrm{add}} > N_{\mathrm{max}}$, we can discard the partial generating function for that configuration, and of course the configuration itself, because it would not make a contribution to the polygon count up to the perimeter lengths we are trying to obtain.

Inspired by Knuth's algorithm for the enumeration of polyominoes [20], we implemented a couple of further enhancements to our SAP algorithm. The first improvement is a superior memory management. A given boundary line configuration does not contribute until order $N = N_{\mathrm{cur}} + N_{\mathrm{add}}$, so we need only retain the first $(N_{\mathrm{max}} - N)/2$ terms in the associated generating function, the factor of 2 arising since every other term is identically 0. In our case the maximum in memory consumption occurred at $W = 24$, where there are approximately 8.1×10^8 distinct configurations and a total of about 2.1×10^9 non-zero terms in the generating functions. So on average there is only about 2.5 non-zero terms per configuration. The second improvement uses a further symmetry of the square lattice. When a column has been completed the configuration is symmetric under reflection. That is the generating functions for the configurations such as, {010122000} and {000112020}, are identical. This symmetry also extends to the touching of the upper/lower borders of the rectangle.

The generalization of the algorithm to calculations of area-weighted moments and the radius of gyration is described in [16]. Note that the additional symmetry mentioned above does not extend to the radius of gyration calculation.

2.2. Parallelization

The computational complexity of the FLM grows exponentially with the number of terms one wishes to calculate. It is therefore little wonder that implementations of the algorithms have always been geared towards using the most powerful computers available. In the past decade or so parallel computing has become the paradigm for high-performance computing. The early machines were largely dedicated MPP machines which more recently have been superceded by clusters.

The transfer-matrix algorithms used in the calculations of the finite lattice contributions are eminently suited for parallel computations.

The most basic concerns in any efficient parallel algorithm is to minimize the communication between processors, and ensure that each processor does the same amount of work and uses the same amount of memory. In practice, one naturally has to strike some compromise and accept a certain degree of variation across the processors.

One of the main ways of achieving a good parallel algorithm using data decomposition is to try to find an invariant under the operation of the updating rules. That is we seek to find some property about the configurations along the boundary line which does not alter in a single iteration. The algorithm for the enumeration of polygons is quite complicated since not all possible configurations occur due to pruning, and an update at a given set of edges might change the state of an edge far removed, for example, when two lower loop ends are joined we have to relabel one of the associated upper loop ends as a lower loop end in the new configuration (see figure 2). However, there is still an invariant, since any edge not directly involved in the update cannot change from being empty to being occupied and vice versa. That is only the kink edges can change their occupation status. This invariant allows us to parallelize the algorithm in such a way that we can do the calculation completely independently on each

processor with just two redistributions of the dataset each time an extra column is added to the lattice.

The main points of the algorithm are summarized below.

1. With the boundary line straight (having no kinks) distribute the data across processors so that configurations with the same occupation pattern along the *lower* half of the boundary line are placed on the same processor.
2. Do the TM update inserting the top-half of a new column. This can be done *independently* by each processor because the occupation pattern in the lower half remains unchanged.
3. Upon reaching the half-way mark, redistribute the data so that configurations with the same occupation pattern along the *upper* half of the boundary line are placed on the same processor.
4. Do the TM update inserting the bottom-half of a new column.
5. Go back to 1.

The redistribution among processors is done as follows:

1. On each processor run through the configurations to establish the configuration pattern c of each configuration and calculate, $n(c)$, the number of configurations with a given pattern.
2. On processor 0 calculate the *global* sum of $n(c)$.
3. Sort the global sum $n(c)$ on processor 0.
4. On processor 0 assign each pattern to a processor $p(c)$ such that:
   (a) Set $p_{\mathrm{id}} = 0$.
   (b) Assign the *most* frequent unassigned pattern c to processor p_{id}.
   (c) If the number of configurations assigned to p_{id} is less than the number of configurations assigned to processor 0, then assign the *least* frequent unassigned patterns to p_{id} until the desired inequality is achieved.
   (d) Set $p_{\mathrm{id}} = (p_{\mathrm{id}} + 1) \bmod N_p$, where N_p is the number of processors.
   (e) Repeat from (b) until all patterns have been assigned.
5. Send $p(c)$ to all processors.
6. On each processor run through the configurations sending each configuration to its assigned processor.

The bulk of the calculations were performed on the facilities of the Australian Partnership for Advanced Computing (APAC). The APAC facility is a Compaq server cluster with 125 ES45's each with four 1 Ghz chips for a total of 500 processors in the compute partition. The cluster has a total peak speed over 1Tflop. Each server node has at least 4 Gb of memory. Nodes are interconnected by a fat-tree low latency (MPI $<$ 5 us), high bandwidth (250 Mb s^{-1} bidirectional) Quadrics network.

In table 1 we have listed the time and memory use of the algorithm for $N_{\max} = 98$ at $W = 22$, using from 1 to 64 processors. The memory use of the single processor job was about 3 Gb. As can be seen the algorithm scales perfectly from 1 to 64 processors, since the total combined CPU time (column 2, format is hours:minutes) used by all processors stays almost constant. Likewise, the elapsed time (column 3, format is hours:minutes:seconds), which is the running time of the program, is halved when the number of processors is doubled. We expect that the rather surprising drop in CPU time at 32 or 64 processors is caused by better single processor optimization by the compiler. One would, for example, expect that the average time taken to fetch elements from main memory drops as the memory size on each individual processor drops from 3 Gb for the computation using a single processor to just under 50 Mb for the 64 processor computation. Another main issue in parallel computing

Table 1. CPU time and memory use for the parallel algorithm for enumerating polygons of maximal perimeter 98 at width 22.

Proc.	CPU time	Elapsed time	Max conf	Min conf	Max term	Min term
1	33:26	33:34:30	94 858 092		202 124 719	
2	34:58	17:31:09	45 332 715	45 312 242	99 729 074	99 050 619
4	34:15	8:35:57	22 762 665	22 667 218	51 880 015	51 263 646
8	34:03	4:16:51	11 692 292	11 525 456	26 498 730	26 097 260
16	34:16	2:09:40	5 880 705	5 707 628	13 523 912	13 037 482
32	33:15	1:03:04	2 941 787	2 821 055	6 934 653	6 451 282
64	32:29	31:07	1 489 116	1 398 768	3 519 013	3 222 199

is that of load balancing, that is, we wish to ensure to the greatest possible extent that the workload is shared equally among all the processors. This aspect is examined via the numbers in columns 4–7. At any given time during the calculation each processor handles a subset of the total number of configurations. For each processor we monitor the maximal number of configurations and terms retained in the generating functions. The load balancing can be roughly gauged by looking at the largest (Max Conf) and smallest (Min Conf) maximal number of configurations handled by individual processors during the execution of the program. In columns 6 and 7 are listed the largest (Max Term) and smallest (Min Term) number of terms retained in the generating functions associated with the subset of configurations. As can be seen the algorithm is quite well balanced. Even with 64 processors, where each processor uses only about 50 Mb of memory, the difference between the processor handling the maximal and minimal number of configurations is less than 10%. The same holds true for the total number of terms retained in the generating functions.

A simple timing of the various subroutines of the parallel algorithm shows that the typical time to do a redistribution is the same as the average time taken per iteration to move the kink once. Since the maximal time use at $N_{\max} = 110$ occurs at $W = 24$, there are 24 iterations and just 2 redistributions per added column, so the overall cost of parallel execution is smaller than 10%.

2.3. Further details

Finally a few remarks of a more technical nature. The number of contributing configurations becomes very sparse in the total set of possible states along the boundary line, and as is standard in such cases one uses a hash-addressing scheme. Since the integer coefficients occurring in the series expansion become very large, the calculation was performed using modular arithmetic [19]. This involves performing the calculation modulo various integers p_i and then reconstructing the full integer coefficients at the end. The p_i are called moduli and must be chosen so they are mutually prime, for example, none of the p_i has a common divisor. The Chinese remainder theorem ensures that any integer has a unique representation in terms of residues. If the largest absolute values occurring in the final expansion is m, then we have to use a number of moduli k such that $p_1 p_2 \cdots p_k/2 > m$. Since we are using a heavily loaded shared facility, the CPU time was more of an immediate limitation than memory and, secondly, more memory was used for the data required to specify the configuration and manage the storage than to store the actual terms of the generating functions. So we used the moduli $p_0 = 2^{62}$, $p_1 = 2^{62} - 1$ and $p_2 = 2^{62} - 3$, which allowed us to represent p_n correctly using these three moduli. The calculation of the area-weighted moments and the radius of gyration requires a lot more memory for the generating functions (plus the radius of gyration

Table 2. The number, p_n, of embeddings of n-step polygons on the square lattice. Only non-zero terms are listed.

n	p_n
92	3959 306 049 439 766 117 380 237 943 449 096
94	26 117 050 944 268 596 220 897 591 868 398 452
96	172 472 018 113 289 556 124 895 798 382 016 316
98	1140 203 722 938 033 441 542 255 979 068 861 816
100	7545 649 677 448 506 970 646 886 033 356 862 162
102	49 985 425 311 177 130 573 540 712 929 060 556 804
104	331 440 783 010 043 009 106 782 321 492 277 936 522
106	2199 725 502 650 970 871 182 263 620 080 571 090 156
108	14 612 216 410 979 678 692 651 320 184 958 285 074 180
110	97 148 177 367 657 853 074 723 038 687 712 338 567 772

calculation involves multiplication with quite large integers), so in this case we used prime numbers of the form $2^{30} - r_i$ for the moduli p_i. Up to 6 primes were needed to represent the coefficients correctly.

Combining all the memory minimization tricks mentioned above allows us to extend the series for the square lattice polygon generating function from 90 terms to 110 terms using at most 36 Gb of memory. The calculations requiring the most resource were at widths 23–25. These cases were done using 40 processors and took about 8–10 h each per prime. The total CPU time required was about 1500 h per prime. Obviously the calculation for each width and prime is totally independent and several calculations can be done simultaneously. A similar total amount of resources was required to calculate the area-moments and the radius of gyration.

In table 2 we have listed the new terms obtained in this work for the number of polygons with perimeter 92–110. The number of polygons of length $\leqslant 56$ can be found in [13], while those up to length 90 are listed in [18].

3. Analysis of the series

To obtain the singularity structure of the generating functions we used the numerical method of differential approximants [12]. Since all odd terms in the series are zero and the first non-zero term is p_4, we actually analysed the function $P(u) = \sum_n p_{2n+4} u^n$, and so on. These functions have critical points at $u = u_c^2$ with the same exponents as those of (2). Our main objective is to obtain very accurate estimates for the connective constant μ and the critical exponents α and ν. In particular, we test a conjecture [3] for the exact value of the connective constant and confirm to a very high degree of precision the exact values of the exponents.

Once the exact values of the exponents have been confirmed, we will turn our attention to the 'fine structure' of the asymptotic form of the coefficients. In particular, we are interested in obtaining accurate estimates for the amplitudes B, D and $E^{(k)}$. We do this by fitting the coefficients to the assumed form (1). In this case our main aim is to test the validity of the predictions for the amplitude combinations mentioned in introduction.

3.1. The polygon generating function

In table 3 we have listed estimates for the critical point u_c^2 and exponent $2 - \alpha$ of the series for the square lattice SAP generating function. The estimates were obtained by averaging

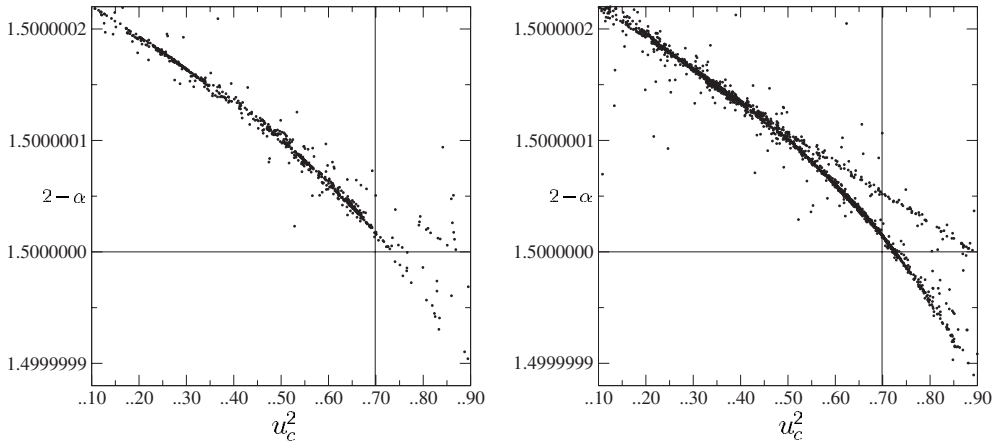

Figure 3. Estimates for the critical exponent $2-\alpha$ versus estimates for the critical point u_c^2 of the square lattice polygon generating function. Each tick label along the x-axis is preceded by the value 0.143 680 6292. The straight lines correspond to $2-\alpha = 3/2$ and $u_c^2 = 0.143\,680\,629\,269\,8685\ldots$.

Table 3. Estimates for the critical point u_c^2 and exponent $2-\alpha$ obtained from second and third order differential approximants to the series for square lattice polygon generating function. L is the order of the inhomogeneous polynomial.

	Second order DA		Third order DA	
L	u_c^2	$2-\alpha$	u_c^2	$2-\alpha$
0	0.143 680 629 242(28)	1.500 000 116(94)	0.143 680 629 246(22)	1.500 000 105(73)
2	0.143 680 629 245(15)	1.500 000 111(63)	0.143 680 629 247(21)	1.500 000 097(81)
4	0.143 680 629 246(16)	1.500 000 107(62)	0.143 680 629 251(22)	1.500 000 080(99)
6	0.143 680 629 250(17)	1.500 000 094(65)	0.143 680 629 244(22)	1.500 000 109(72)
8	0.143 680 629 249(22)	1.500 000 094(72)	0.143 680 629 249(28)	1.500 000 09(14)
10	0.143 680 629 248(19)	1.500 000 095(66)	0.143 680 629 252(28)	1.500 000 06(15)
12	0.143 680 629 246(21)	1.500 000 105(70)	0.143 680 629 247(18)	1.500 000 100(70)
14	0.143 680 629 242(20)	1.500 000 116(66)	0.143 680 629 245(26)	1.500 000 099(99)
16	0.143 680 629 252(18)	1.500 000 086(63)	0.143 680 629 247(25)	1.500 000 097(94)
18	0.143 680 629 254(15)	1.500 000 076(65)	0.143 680 629 247(22)	1.500 000 098(81)
20	0.143 680 629 238(26)	1.500 000 122(74)	0.143 680 629 242(23)	1.500 000 113(87)

values obtained from second and third order differential approximants. For each order L of the inhomogeneous polynomial we averaged over those approximants to the series which used at least the first 45 terms of the series (that is, polygons of perimeter at least 90). The error quoted for these estimates reflects the spread (basically one standard deviation) among the approximants. Note that these error bounds should *not* be viewed as a measure of the true error as they cannot include possible systematic sources of error. Based on these estimates we conclude that $u_c^2 = 0.143\,680\,629\,25(5)$ and $\alpha = 0.500\,0001(2)$. This analysis adds strongly to the already very convincing evidence that the critical exponent $\alpha = 1/2$ exactly.

If we take the conjecture $\alpha = 1/2$ to be true, we can obtain a refined estimate for the critical point u_c^2 enabling us to check whether or not the estimates for u_c^2 still agree with the root of the polynomial. In figure 3 we have plotted estimates for the critical exponent $2-\alpha$ against estimates for the critical point u_c^2. Each dot in the left (right) panel of this figure represents a pair of estimates obtained from a second (third) order inhomogeneous

Table 4. Estimates for the critical point u_c^2 and exponents $-(\alpha + 2\nu)$ and $2 - (\alpha + 2\nu)$ obtained from second (top-half) and third (bottom-half) order differential approximants to the series for the radius of gyration and first area-weighted moment of square lattice SAP.

Series:	$\mathcal{R}_g^2(u)$		$\mathcal{P}^{(1)}(u)$	
L	u_c^2	$2-\alpha$	u_c^2	$2-\alpha$
0	0.143 680 5865(92)	−1.999 681(36)	0.143 680 6053(50)	0.000 122(15)
2	0.143 680 592(10)	−1.999 704(43)	0.143 680 609(10)	0.000 143(60)
4	0.143 680 5889(82)	−1.999 689(35)	0.143 680 609(11)	0.000 139(50)
6	0.143 680 583(23)	−1.999 676(82)	0.143 680 604(12)	0.000 07(16)
8	0.143 680 588(10)	−1.999 680(55)	0.143 680 608(10)	0.000 10(10)
10	0.143 680 591(12)	−1.999 703(59)	0.143 680 616(22)	−0.000 11(79)
0	0.143 680 6081(85)	−1.999 822(53)	0.143 680 607(12)	0.000 21(28)
2	0.143 680 605(13)	−1.999 803(79)	0.143 680 616(11)	0.000 11(12)
4	0.143 680 6074(92)	−1.999 812(61)	0.143 680 6143(73)	0.000 108(42)
6	0.143 680 606(11)	−1.999 817(71)	0.143 680 6166(64)	0.000 095(26)
8	0.143 680 6057(93)	−1.999 809(53)	0.143 680 6148(45)	0.000 083(40)
10	0.143 680 606(11)	−1.999 817(61)	0.143 680 6154(55)	0.000 099(21)

differential approximant. The order of the inhomogeneous polynomial was varied from 0 to 10. As can be seen the estimates for the critical exponent cross the line $2 - \alpha = 3/2$ at a value $u_c^2 \simeq 0.143\,680\,629\,273$, which is slightly larger than the value obtained from the root of the polynomial suggested as possibly providing the exact value. So this is the first direct evidence that the suggestion could be wrong. Since the difference only occurs in the 12th significant digit, we do not feel confident that the numerical evidence alone is sufficient to settle the matter. It may be the case that there are subtle systematic trends in the estimates, which preclude them from having converged to the true values of the parameters. However, as emphasized in [18] the other zero of the polynomial is at $u_c^2 = -0.155\,7288\ldots$, and as was the case in this previous analysis, we see no evidence of such a singularity, which casts serious doubt on the validity of the suggestion. Particularly, since we are not aware of any arguments as to why we might not expect to see the singularity on the negative real axis from our series analysis. Ultimately, we will let the reader make their own judgment.

Based on this analysis we adopt the value $u_c^2 = 0.143\,680\,629\,273(3)$, and thus $\mu = 2.638\,158\,530\,31(3)$ as our final estimates.

3.2. The radius of gyration and area-weighted moments

Table 4 contains estimates for u_c^2 and the critical exponents of the generating functions (2) for the radius of gyration and first area-weighted moment. Suffice to say, the estimates of the exponents are in agreement with the conjectured exact value $\nu = 3/4$.

3.3. The amplitudes

The asymptotic form of the coefficients p_n of the polygon generating function has been studied in detail previously [4, 18]. As argued in [4] there is no sign of non-analytic corrections-to-scaling exponents to the polygon generating function, and one, therefore, finds that

$$p_n = \mu^n n^{-5/2} \sum_{i \geqslant 0} a_i / n^i. \tag{6}$$

This form was confirmed with great accuracy in [18]. Estimates for the leading amplitude $B = a_0$ can thus be obtained by fitting p_n to the form given in equation (6) using increasing

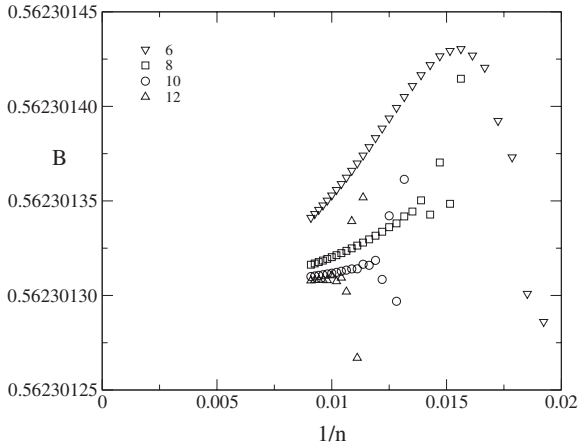

Figure 4. Estimates for the amplitude B versus $1/n$. Each dataset is obtained by fitting p_n to the form (6) using from 6 to 12 correction terms.

values of k. As in [16] we find it useful to check the behaviour of the estimates by plotting the results for the leading amplitude versus $1/n$ (see figure 4), where p_n is the last term used in the fitting, and n is varied from 110 down to 50. We again note that as more and more correction terms are added to the fits, the estimates exhibit less curvature and that the slope becomes less steep. This is very strong evidence that (6) indeed is the correct asymptotic form of p_n. We estimate that $B = 0.562\,301\,30(2)$.

The asymptotic form of the coefficients r_n in the generating function for the radius of gyration was studied in [16]. When fitting to a form similar to equation (6), assuming that there are only analytic corrections-to-scaling, we found that the amplitudes of higher order terms are very large and that the leading amplitude converge rather slowly. This indicates that this asymptotic form is incorrect. We found that the coefficients fit better if we assume a leading non-analytic correction-to-scaling exponent $\Delta = 3/2$. This result confirms the prediction of Nienhuis [25]. Note that since the polygon generating function exponent $2 - \alpha = 3/2$, a correction-to-scaling exponent $\Delta = 3/2$ is perfectly consistent with the asymptotic form (6). Because $2 - \alpha + \Delta$ is an integer, the non-analytic correction term becomes part of the analytic background term [4]. We thus proposed the following asymptotic form:

$$r_n = \mu^n n \left[BD + \sum_{i \geqslant 0} a_i / n^{i/2} \right]. \tag{7}$$

Alternatively, we could fit to the form

$$r_n / p_n = n^{7/2} \left[D + n^{5/2} \sum_{i \geqslant 0} a_i / n^{i/2} \right]. \tag{8}$$

In figure 5 we show the leading amplitudes resulting from such fits while using from 6 to 12 terms in these expansions. Also shown in these figures (solid lines) are the predicted exact value of BD, given in equation 4, and the prediction for D using the estimate for B obtained above. As can be seen the leading amplitudes clearly converge towards their expected values, and from these plots we can conclude that the prediction for BD has been confirmed to at least 5 digit accuracy. Assuming that equation (4) is exact and using the very accurate estimate for B, we find that $D = 0.056\,309\,437(2)$.

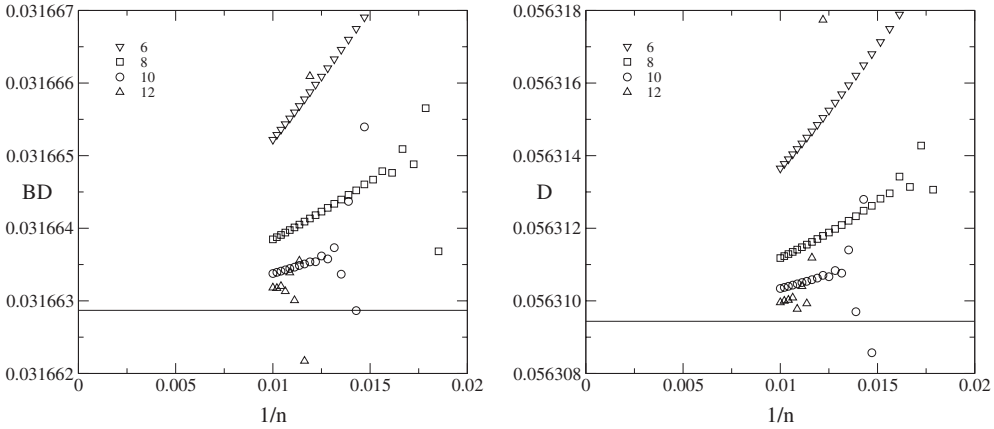

Figure 5. Estimates for the amplitudes BD and D versus $1/n$. Each dataset is obtained by fitting r_n to the form (7) and r_n/p_n to the form (8) while using from 6 to 12 correction terms.

Table 5. Exact values and estimates from square lattice polygons for the universal amplitude combinations.

Amplitude	Exact value	Estimate
E_1	$0.795\,774\,715\ldots \times 10^{-1}$	$0.795\,773(2) \times 10^{-1}$
E_2B	$0.335\,953\,483\ldots \times 10^{-2}$	$0.335\,952(2) \times 10^{-2}$
E_3B^2	$0.100\,253\,732\ldots \times 10^{-3}$	$0.100\,253(1) \times 10^{-3}$
E_4B^3	$0.237\,553\,411\ldots \times 10^{-5}$	$0.237\,552(2) \times 10^{-5}$
E_5B^4	$0.475\,738\,345\ldots \times 10^{-7}$	$0.475\,736(3) \times 10^{-7}$
E_6B^5	$0.836\,630\,215\ldots \times 10^{-9}$	$0.836\,624(5) \times 10^{-9}$
E_7B^6	$0.132\,514\,776\ldots \times 10^{-10}$	$0.132\,514(2) \times 10^{-10}$
E_8B^7	$0.192\,419\,637\ldots \times 10^{-12}$	$0.192\,418(2) \times 10^{-12}$
E_9B^8	$0.259\,465\,635\ldots \times 10^{-14}$	$0.259\,464(2) \times 10^{-14}$
$E_{10}B^9$	$0.328\,063\,262\ldots \times 10^{-16}$	$0.328\,062(4) \times 10^{-16}$

Next we test the predictions [27] for the amplitude combinations of the area-weighted moments:

$$E_{2k}B^{2k-1} = -\frac{c_{2k}}{4\pi^{3k}}\frac{(3k-2)!}{(6k-3)!} \qquad E_{2k+1}B^{2k} = \frac{c_{2k+1}}{(3k)!\pi^{3k+1}2^{6k+2}} \tag{9}$$

for $k \in \mathbb{N}$, where we used the known result that $E_1 = 1/4\pi$ [1] and the amplitudes E_k are related to the amplitudes $E^{(k)}$ in equation (1) by $E_k = E^{(k)}/k!$. The numbers c_k are given by the recursion

$$c_n + (3n-4)c_{n-1} + \frac{1}{2}\sum_{r=1}^{n-1} c_{n-r}c_r = 0 \qquad c_0 = 1. \tag{10}$$

The first few values are $c_1 = 1, c_2 = -5/2$, $c_3 = 15, c_4 = -1105/8, c_5 = 1695, c_6 = -414\,125/16, c_7 = 472\,200, c_8 = -1282\,031\,525/128, c_9 = 242\,183\,775, c_{10} = -1683\,480\,621\,875/256$.

We fit the coefficients to the form

$$a_n^{(k)} \approx k!\mu^n n^{(\alpha+2k\nu)-3}\left[E_k + \sum_{i\geqslant 0} a_i/n^{1+i/2}\right]. \tag{11}$$

We obtain several datasets by varying the number of terms used in this formula from 8 to 12. To obtain the final estimates we do a simple linear regression on the data for the amplitudes as a function of $1/n$ extrapolating to $1/n \to 0$. We estimate the error on the amplitude estimate from the spread among the different datasets. In this way, we obtain the results for the amplitude combinations listed in table 5.

It is clear that the results for the first 10 area-weighted moments are in excellent agreement with the numerical estimates. On this basis we conclude that the conjectured scaling function and derived exact amplitude combinations [27] are correct.

4. Conclusion

We have presented an improved and parallel algorithm for the enumeration of self-avoiding polygons on the square lattice. This algorithm has enabled us to obtain polygons up to perimeter length 110 and their radius of gyration and area-weighted moments up to perimeter 100. Our extended series enables us to give an extremely precise estimate of the connective constant $\mu = 2.638\,158\,530\,31(3)$. This estimate provides the first direct evidence that an earlier conjecture for the exact value of μ could be incorrect. We confirmed to a very high degree of accuracy the predicted exponent values $\alpha = 1/2$ and $\nu = 3/4$. We also obtained very accurate estimates for the leading amplitude $B = 0.562\,301\,30(2)$ of the asymptotic expansion of p_n, and confirmed the correctness of theoretical predictions for the values of the amplitude combinations BD and $E^{(k)}B^{k-1}$.

Acknowledgments

The calculations presented in this paper would not have been possible without a generous grant of computer time on the server cluster of the Australian Partnership for Advanced Computing (APAC). We also used the computational resources of the Victorian Partnership for Advanced Computing (VPAC). We gratefully acknowledge financial support from the Australian Research Council.

References

[1] Cardy J L 1994 Mean area of self-avoiding loops *Phys. Rev. Lett.* **72** 1580–3
[2] Cardy J L and Guttmann A J 1993 Universal amplitude combinations for self-avoiding walks, polygons and trails *J. Phys. A: Math. Gen.* **26** 2485–94
[3] Conway A R, Enting I G and Guttmann A J 1993 Algebraic techniques for enumerating self-avoiding walks on the square lattice *J. Phys. A: Math. Gen.* **26** 1519–34
[4] Conway A R and Guttmann A J 1996 Square lattice self-avoiding walks and corrections to scaling *Phys. Rev. Lett.* **77** 5284–7
[5] Delest M P and Viennot G 1984 Algebraic languages and polyominoes enumeration *Theor. Comput. Sci.* **34** 169–206
[6] Enting I G 1980 Generating functions for enumerating self-avoiding rings on the square lattice *J. Phys. A: Math. Gen.* **13** 3713–22
[7] Enting I G and Guttmann A J 1985 Self-avoiding polygons on the square, L and Manhattan lattices *J. Phys. A: Math. Gen.* **18** 1007–17
[8] Enting I G and Guttmann A J 1990 On the area of square lattice polygons *J. Stat. Phys.* **58** 475–84
[9] Enting I G and Guttmann A J 1992 Self-avoiding rings on the triangular lattice *J. Phys. A: Math. Gen.* **25** 2791–807
[10] Fisher M E 1989 Fractal and nonfractal shapes in two-dimensional vesicles *Physica* D **38** 112–8
[11] Fisher M E, Guttmann A J and Whittington S G 1991 Two-dimensional lattice vesicles and polygons *J. Phys. A: Math. Gen.* **24** 3095–106

[12] Guttmann A J 1989 Asymptotic analysis of power-series expansions in *Phase Transitions and Critical Phenomena vol 13* ed C Domb and J L Lebowitz (New York: Academic) pp 1–234
[13] Guttmann A J and Enting I G 1988 The size and number of rings on the square lattice *J. Phys. A: Math. Gen.* **21** L165–L172
[14] Hiley B J and Sykes M F 1961 Probability of initial ring close in the restricted random-walk model of a macromolecule *J. Chem. Phys.* **34** 1531–7
[15] Hughes B D 1995 *Random Walks and Random Environments, Vol I Random Walks* (Oxford: Clarendon)
[16] Jensen I 2000 Size and area of square lattice polygons *J. Phys. A: Math. Gen.* **33** 3533–43
[17] Jensen I and Guttmann A J 1998 Self-avoiding walks, neighbour-avoiding walks and trails on semiregular lattices *J. Phys. A: Math. Gen.* **30** 8137–45
[18] Jensen I and Guttmann A J 1999 Self-avoiding polygons on the square lattice *J. Phys. A: Math. Gen.* **32** 4867–76
[19] Knuth D E 1969 *Seminumerical Algorithms. The Art of Computer Programming* vol 2 (Reading, MA: Addison-Wesley)
[20] Knuth D E 2001 Polynum and Polyslave the program is available from Knuth's Home-page at http://Sunburn.Stanford.EDU/~knuth/programs.html#polyominoes
[21] Leibler S, Singh R R P and Fisher M E 1987 Thermodynamic behavior of two-dimensional vesicles *Phys. Rev. Lett.* **59** 1989–92
[22] Lin K Y 2000 Universal amplitude combinations for self-avoiding walks and polygons on the honeycomb lattice *Physica* A **275** 197–206
[23] Lin K Y and Kao Y M 1999 Universal amplitude combinations for self-avoiding walks and polygons on directed lattices *J. Phys. A: Math. Gen.* **32** 6927–38
[24] Lin K Y and Lue S J 1999 Universal amplitude combinations for self-avoiding polygons on the kagome lattice *Physica* A **270** 453–61
[25] Nienhuis B 1982 Exact critical point and critical exponents of O(n) models in two dimensions *Phys. Rev. Lett.* **49** 1062–5
[26] Privman V and Rudnick J 1985 Size of rings in two dimensions *J. Phys. A: Math. Gen.* **18** L789–L793
[27] Richard C, Guttmann A J and Jensen I 2001 Scaling function and universal amplitude combinations for self-avoiding polygons *J. Phys. A: Math. Gen.* **34** L495–L501