Exact Percolation Probability on the Square Lattice

Stephan Mertens$^{1,2}$

1 Institut für Physik, Universität Magdeburg, Universitätsplatz 2, 39016 Magdeburg, Germany
2 Santa Fe Institute, 1399 Hyde Park Rd., Santa Fe, NM 87501, USA
E-mail: mertens@ovgu.de

Abstract. We present an algorithm to compute the exact probability $R_n(p)$ for a site percolation cluster to span an $n \times n$ square lattice at occupancy $p$. The algorithm has time and space complexity $O(\lambda^n)$ with $\lambda \approx 2.6$. It allows us to compute $R_n(p)$ up to $n = 24$. We use the data to compute estimates for the percolation threshold $p_c$ that are several orders of magnitude more precise than estimates based on Monte-Carlo simulations.

1. Introduction

Most of the work in percolation is intimately connected with computer simulations. Over the years, various algorithms have been developed [1]. Think of the Hoshen-Kopelman algorithm to identify clusters [2], the cluster growth algorithms [3, 4] or the cluster hull generating algorithm [5, 6, 7] to name but a few. Another example is the invasion percolation algorithm, that works well independently of the spatial dimension [8] and even on hyperbolic lattices with their “infinite” dimension [9].

The microcanonical union-find algorithm by Newman and Ziff [10, 11] was a major improvement for Monte-Carlo simulations. It is not only exceedingly efficient, but also simple and beautiful.

In this contribution we will introduce an algorithm for the exact enumeration of percolating configurations in 2d lattices. We will discuss this algorithm for site percolation on the square lattice, but the method can easily be adapted to other 2d lattices.

Our motivation to devise such an algorithm and to run it is twofold. First, because we think that it is always good to have exact solutions, even for small systems. And second, because one can use the enumeration results to compute estimates for the percolation threshold $p_c$ that are more accurate than anything achievable with Monte Carlo simulations.

The latter is a recent trend in percolation. Exact estimators for small systems combined with careful extrapolations have lead to estimates for $p_c$ with unprecedented accuracy for many different lattices, see [12] and references therein.

The situation for site percolation is depicted in Table 1. TM refers to transfer matrix methods as a label for enumerative, exact solutions of small systems combined with careful extrapolations. MC stands for Monte-Carlo results. As you can see, in 2008 TM took over MC. We will also use TM and add two more values to this Table.
The paper is organized in three sections plus conclusions and appendix. The first section defines the task and introduces the generating function. The second section comprises a detailed description of the algorithm, followed by a brief discussion of the results. In the third section we describe the computation of high precision values for the percolation threshold $p_c$. Sections two and three can be read independently. The appendix explains some of the combinatorics behind the algorithm.

2. The Number of Percolating Configurations

Our focus is on site percolation on the $n \times m$ square lattice, where percolation refers to spanning the $m$ direction. In particular we want to compute the number $A_{n,m}(k)$ of spanning configurations with $k$ sites occupied. These numbers are conveniently organized in terms of the generating function

$$F_{n,m}(z) \equiv \sum_{k=0}^{nm} A_{n,m}(k) z^k.$$  \hspace{1cm} (1)

The spanning probability at occupancy $p$ is given by

$$R_{n,m}(p) = (1-p)^{nm} F_{n,m} \left( \frac{p}{1-p} \right).$$ \hspace{1cm} (2)

The numbers $A_{n,m}(k)$ can be computed by enumerating all $2^{nm}$ configurations. Enumerating only the hulls of the spanning clusters \cite{6} reduces this complexity to $O(1.7^{nm})$, but even with this speedup, the maximum feasible system size is $8 \times 8$ \cite{25}.

Obviously $A_{n,m}(k) = 0$ for $k < m$. For $k = m$, the only spanning cluster is a straight line of length $m$,

$$A_{n,m}(m) = n.$$ \hspace{1cm} (3a)

For $k = m + 1$ the spanning cluster is either a straight line with one extra site which
does not contribute to spanning, or it is a line with a kink:

$$A_{n,m}(m + 1) = n \binom{nm - m}{1} + 2(n - 1)(m - 2) \quad (m \geq 2).$$  \hfill (3b)

For $k = m + 2$ a spanning configuration is either a straight line with two free sites, a line with one kink and one free site or a line with two small or one large kink:

$$A_{n,m}(m + 2) = n \binom{nm - m}{2} + 2(n - 2) \left( \binom{m - 2}{2} + (m - 2) \right)$$
$$+ 2(n - 1) \left( (nm - m - 3) + (m - 3)(nm - m - 2) \right) \quad (m \geq 2). \hfill (3c)$$

The formulae for $k = m + 3, m + 4, \ldots$ can be derived using similar considerations, but they quickly get complicated and are not very illuminating. And of course we are here to have the computer do this work.

In the high density regime we focus on the number of empty sites. Obviously we need at least $n$ empty sites to prevent spanning,

$$A_{n,m}(nm - k) = \binom{nm}{mn - k} \quad (k < n) \hfill (4a)$$

$n$ empty sites can block spanning only if they form a path that a king can take from the left column of an $n \times m$ chess board to the right column in $n - 1$ moves. Let $I_m(n)$ denote the number of such paths. Then

$$A_{n,m}(nm - n) = I_m(n). \hfill (4b)$$

This number of royal paths would be $m3^{n-1}$ if the king were allowed to step on rows above and below the board. If he cannot leave the board, the number of paths is given by a more intricate formula \[26\]. On a quadratic board, however, the number of paths can be expressed by the Motzkin numbers $M_k$ [A.3]:

$$I_m(n) = (n + 2)3^{n-2} + 2 \sum_{k=0}^{n-3} (n-k-2)3^{n-k-3}M_k. \hfill (5)$$

Equations \[3\] and \[4\] can be used as sanity check for the algorithm. Another sanity check is provided by a surprising parity property [27],

$$F_{n,m}(-1) = \sum_{\text{even } k} A_{n,m}(k) - \sum_{\text{odd } k} A_{n,m}(k) = (-1)^s(n,m) \hfill (6a)$$

with

$$s(n, m) = \left\lfloor \frac{m}{2} \right\rfloor n + \left\lceil \frac{m}{2} \right\rceil. \hfill (6b)$$

3. Algorithm

If you want to devise an efficient algorithm for a computational problem, you would be well advised to consult one of the approved design principles of the algorithms craft \[28\]. In our case the algorithmic paradigm of choice is **dynamic programming**. The idea of dynamic programming is to identify a collection of subproblems and tackling them one by one, smallest first, using the solutions of the small problems to help figure out larger ones until the original problem is solved.
3.1. Dynamic Programming and Transfer Matrix

In the present case, a natural set of subproblems is the computation of $F_{n,m}(z)$ for a given occupation pattern in the $m$th row. For $n = 2$, for example, either the left side, the right side or both sites are occupied:

$$F_{2,m}(z) = F_{2,m}^l(z) + F_{2,m}^r(z) + F_{2,m}^{lr}.$$  

An all empty row can not occur in spanning configurations. The $m$th row can only represent a spanning configuration if the $(m-1)$th row is either $r$ or $r^l$, hence

$$F_{2,m}^r(z) = z F_{2,m-1}^r(z) + z F_{2,m-1}^{lr}(z).$$

Complementing the equations for $F_{2,m}^r(z)$ and $F_{2,m}^{lr}(z)$, we can write this as matrix-vector product

$$
\begin{pmatrix}
F_{2,m}^r(z) \\
F_{2,m}^l(z) \\
F_{2,m}^{lr}(z)
\end{pmatrix}
=
\begin{pmatrix}
z & 0 & z \\
0 & z & z \\
z^2 & z^2 & z^2
\end{pmatrix}
\begin{pmatrix}
F_{2,m-1}^r(z) \\
F_{2,m-1}^l(z) \\
F_{2,m-1}^{lr}(z)
\end{pmatrix}.
$$ (7)

For general values of $n$ we can write

$$F_{n,m}^\sigma(z) = \sum_{\sigma'} T_{\sigma,\sigma'}(z) F_{n,m-1}^{\sigma'}(z),$$ (8)

where $\sigma$ and $\sigma'$ denote possible configurations that can occur in a row of width $n$. Each $\sigma$ in row $m$ represents all compatible configurations in the $n \times m$ lattice “above” it. Appropriately we refer to $\sigma$ as signature of all these configurations.

The idea of the algorithm to compute $F_{n,1}^\sigma(z)$ for all signatures $\sigma$ and then use (8) to iteratively compute $F_{n,2}^\sigma(z)$, $F_{n,3}^\sigma(z)$, etc., thereby reducing the two-dimensional problem with complexity $O(2^{nm})$ to a sequence of $m$ one-dimensional problems of complexity $O(\lambda^n)$ each.

This particular form of a dynamic programming algorithm is known as transfer matrix method. When transfer matrices were introduced into statistical physics by Kramers and Wannier in 1941 [29], they were meant as a tool for analytical rather than numerical computations. Their most famous application was Onsager’s solution of the two dimensional Ising model in 1944 [30]. With the advent of computers, transfer matrices were quickly adopted for numerical computations, and already in 1980, they were considered “an old tool in statistical mechanics” [31]. Yet the idea to use transfer matrices to compute spanning probabilities in percolation emerged only recently, and not in statistical physics but in theoretical computer science. For their analysis of a two-player board game, Yang, Zhou and Li [23] developed a dynamic programming algorithm to compute numerically exact values of the spanning probability for a given value of the occupation probability $p$. We build on their ideas for the algorithm presented here.

3.2. Signatures

Signatures are the essential ingredients of the algorithm. How do we code them? And what is their number? Both questions can be answered by relating signatures to balanced strings of parentheses.
Occupied sites in a signature can belong to the same cluster. Either because they are neighbors within the signature or because they are connected through a path in the lattice beyond the signature. In any case the set of signature sites that belong to the same cluster has a unique left to right order. This allows us to encode cluster sites using left and right parentheses (Figure 1). We represent the leftmost site of a cluster by two left parentheses ((, the rightmost site by two right parentheses )) and any intermediate cluster site by )(. An occupied site that is not connected to any other site within the signature is represented by ()

The resulting string is a balanced parentheses expression (with some white spaces from unoccupied sites), i.e., the total number of left parentheses equals the total number right parentheses, and counting from left to right, the number of right parentheses never exceeds the number of left parentheses. The latter is due to the topological constraints in two dimensions (two clusters can not cross). Identifying all sites of a cluster in a signature means matching parentheses in a balanced string of parentheses, a task that can obviously be accomplished in time $O(n)$.

Since we take into account only configurations that contain at least one spanning cluster, we need to identify clusters that are connected to the top row. We could follow [23] and use brackets [], [], and [] to represent those clusters in a signature, thereby maintaining the concept of balanced strings of parentheses. It is more efficient, however, to use only a single symbol like || for sites that are connected to the top. We can think of all || sites being connected to each other by an all occupied zeroth row. Identifying the left- and rightmost || site in a signature can again be done in time $O(n)$.

The total number of signatures for spanning configurations depend on the width $n$ and the depth $m$. Some signatures can occur only in lattices of a minimum depth. The most extreme case in this respect are nested arcs, represented by signatures like

\[( ( \cdots )) \]

which can only occur if $m \geq \lceil n/2 \rceil$. If $m$ is larger than this, however, the total number of signatures will only depend on $n$. In the Appendix we will prove that this number is given by

\[ S_n = \sum_{r=0}^{\lfloor n/2 \rfloor} \binom{n+1}{2r+2} \binom{2r+2}{r} \frac{3}{r+3} \equiv M_{n+1,2}, \] (9)
Exact Percolation Probability on the Square Lattice

Figure 2. Increasing the system size by adding a new site in the \((c + 1)\)th column of the partially filled row. The state of the new site may affect the cluster structure of the whole lattice, but we only need to update the signature, the cluster structure as it shows in the \(n\) sites colored gray.

where \(M_{n+1,2}\) is the second column in the Motzkin triangle \([A.8]\). The first terms of the series \(S_1, S_2, \ldots\) are

\[
1, 3, 9, 25, 69, 189, 1422, 3915, 10813, 29964, 83304, 232323, 649845, \ldots
\]

We also show in the Appendix, that \(S_n\) grows asymptotically as

\[
S_n \sim \frac{\sqrt{3} 3^{n+3}}{\sqrt{4\pi n^2}} \left(1 - \frac{159}{16n} + \frac{36505}{512n^2} + O\left(\frac{1}{n^3}\right)\right). \quad (10)
\]

3.3. From one dimension to zero dimensions

The central idea of the transfer matrix method \([3]\) is to transform a two dimensional problem into a sequence of one dimensional problems, as a result of which the complexity reduces from \(O(2^{mn})\) to \(O(m2^n S_n)\). This idea can be applied again. By building the new row by site (Figure 2), we can subdivide the one dimensional problem into a sequence of \(n\) zero dimensional problems. This reduces the complexity from \(O(m2^n S_n)\) to \(O(nmS_n)\).

In this approach, the new row is partially filled up to column \(c\). The corresponding signatures contain a kink at column \(c\). Each signature is mapped to two signatures, depending on whether the added site at columns \(c + 1\) is empty or occupied (Figure 2).

The number of signatures \(S_{n,c}\) depends on \(n\) and \(c\). We can think of a partially filled row as a full row of width \(n + 1\), where an extra site has been inserted between positions \(c\) and \(c + 1\), this extra site being empty in all signatures. This gets us

\[
S_n \leq S_{n,c} \leq S_{n+1}, \quad (11)
\]

where the first inequality reflects the fact that the site at position \(c\) has only two neighbors (left and up), and therefore less restrictions to take on “legal” values. The actual value of \(S_{n,c}\) depends on \(c\), see Table [2].
Figure 3. Signature of the cluster configuration of Fig 2 (top), and signatures after adding a new site that is either empty (middle) or occupied (bottom).

| n \ c | 1  2  3  4  5  6  7  8 |
|------|----------------------|
| 1    | 1                    |
| 2    | 4  3                 |
| 3    | 12 11 9              |
| 4    | 33 31 31 25          |
| 5    | 91 85 87 85 69       |
| 6    | 249 233 237 237 233 189 |
| 7    | 683 638 651 646 651 638 518 |
| 8    | 1877 1751 1788 1777 1787 1752 1422 |

Table 2. Values of $S_{n,c}$.

Figure 4. Number of signatures $S_{n,c^*}$ (plain) and $\tilde{S}_{n,c^*}$ (condensed). The lines are $S_{n,c^*} \approx 0.350 \cdot 2.85^n$ and $\tilde{S}_{n,c^*} \approx 0.196 \cdot 2.61^n$. 
Table 3. Extension rules of the transfer matrix algorithm. Each cell shows the resulting configuration if the added site is empty (top) or occupied (bottom). Some update rules are non-local: a $\|$ may only be deleted, if there is at least one other $\|$ elsewhere in the signature, b deleting $((\ )$ or $((\ ))$ means shortening of the corresponding cluster, c the whole cluster has to be connected to the top, and d two clusters merge. Configurations with $E$ must never occur.

3.4. Condensation

The number of signatures is paramount for the complexity of the algorithm. Let $S_{n,c^*}^\star \equiv \max_c S_{n,c}^\star$. Figure 4 shows that $S_{n,c^*}^\star \approx 0.350 \cdot 2.85^n$. In particular, $S_{n,c^*}^\star$ exceeds the “Giga” threshold ($10^9$) for $n > 20$. This is significant because we need to store pairs $(\sigma, F^\sigma)$ for all signatures. For $S_{n,c^*}^\star$ above $10^9$, this means hundreds of GBytes of RAM. In this regime, space (memory) rather than time will become the bottleneck of our algorithm.

It turns out, however, that we can reduce the number of signatures. Consider a signature in row $m$ that contains the pattern $\ldots () \ldots$. All signatures in row $m+1$ that are derived from it, do not depend on whether the single () is actually occupied. Hence we can identify all signatures $\sigma = \ldots () \ldots$ with signatures $\sigma' = \ldots () \ldots$ and store only one pair $(\sigma, F^\sigma)$ instead of two pairs $(\sigma, F^\sigma)$ and $(\sigma', F'^{\sigma'})$. For the same reason, we can also identify $\ldots ((() \ldots$ with $\ldots ((() \ldots$ and $\ldots ((() \ldots$ with $\ldots ((() \ldots$ and store only one of each pair.

In addition, the symmetry of the problem allows us to identify each signature of a full row $(c = n)$ with its mirror image.

Let $\tilde{S}_{n,c}$ denote the number of signatures that have been condensed by these rules, and let $\tilde{S}_{n,c^*} = \max_c \tilde{S}_{n,c}$. Figure 4 shows that $\tilde{S}_{n,c^*} \approx 0.196 \cdot 2.61^n$. The number of condensed signatures exceeds the “Giga” threshold only for $n > 23$. 
3.5. Implementation

The elementary step of the transfer matrix algorithm is the addition of a new lattice site in a partially filled row (Figure 3). This is implemented in a subroutine \( \text{extend}(\sigma,c) \) that takes a signature \( \sigma \) and a column \( c \) as input and returns a pair \( \sigma_0, \sigma_1 \), where \( \sigma_0 \) (\( \sigma_1 \)) follows from \( \sigma \) by adding an empty (occupied) site in column \( c \). The rules for these computations are depicted in Table 3. The signatures \( \sigma_0 \) and \( \sigma_1 \) are of course condensed before they are returned. The subroutine \( \text{extend} \) is then called from within a loop over rows and columns of the lattice (Figure 5).

The algorithm maintains two lists \( L_{\text{old}} \) and \( L_{\text{new}} \) of configurations, where a configuration consists of a signature \( \sigma \) and the corresponding generating function \( F^\sigma \). \( L_{\text{old}} \) is initialized with the \( 2^n - 1 \) first row configurations, in which each occupied site is connected to the top row by definition. \( L_{\text{new}} \) is initially empty.

A configuration \( (\sigma,F^\sigma) \) is removed from \( L_{\text{old}} \) and \( \sigma_0, \sigma_1 = \text{extend}(\sigma,c) \) is computed. Then the configurations \( (\sigma_0,F^\sigma) \) and \( (\sigma_1,F^\sigma) \) are inserted into \( L_{\text{new}} \). This is repeated until \( L_{\text{old}} \) is empty. At this point, \( L_{\text{old}} \) and \( L_{\text{new}} \) swap their names, and the process can be iterated to add the next site to the lattice.

If the lattice has reached its final size \( nm \), the total generating function is the sum over all \( F^\sigma \) for \( (\sigma,F^\sigma) \in L_{\text{old}} \).

With its six symbols \( \lfloor \rceil,\lfloor(\rceil,\rfloor) \) and \( () \), a signature can be interpreted as a senary number with \( n \) digits. For \( n \leq 24 \), this number always fits into a 64-bit integer. Since

\[
2^{64} = 3,520,522,010,102,100,444,244,424,6
\]

we can also represent signatures of size \( n = 25 \) by 64-bit integers, provided we choose the senary representation wisely: \( ( ) = 4_6 \) and \( () = 5_6 \).

The lists \( L \) maintained by the algorithm contain up to \( S_{n,c} \) configurations. An efficient data structure is mandatory. We take advantage of the fact that the signatures
can be ordered according to their senary value. This allows us to use an ordered associative container like `set` or `map` from the standard C++ library. Those guarantee logarithmic complexity $O(\log S_{n,c})$ for search, insert and delete operations. \[32\]

Equipped with a compact representation for the signatures and an efficient data structure for the list of configurations, we still face the challenge to store the large coefficients $A_{n,m}(k)$ of the generating functions $F_{n,m}$. These numbers quickly get too large to fit within the standard fixed width integers with 32, 64 or 128 bits (Figure 6).

One could use multiple precision libraries to deal with this problem, but it is much more efficient to stick with fixed width integers and use modular arithmetic: with $b$-bit integers, we compute $U_{n,m}^{(i)}(k) \equiv A_{n,m}(k) \mod p_i$ for a set of prime moduli $p_i < 2^b$ such that $\prod p_i > \max_k A_{n,m}(k)$. We can then use the Chinese Remainder Theorem to recover $A_{n,m}(k)$ from the $U_{n,m}^{(i)}(k)$. This way we can trade space (bit size of numbers) for time (one run for each prime modulus).

3.6. Performance

Figure 7 shows the actual memory consumption to compute $F_{n,n}$ with $b$-bit integers. Note that data structures like red-black-trees require extra data per stored element. Unfortunately the product of all primes below $2^8$ is not large enough to compute the coefficients in $F_{n,n}$ via modular arithmetic and the Chinese Remainder Theorem if $n \geq 19$. Hence 8-bit integers are essentially useless.

Extrapolation of the data in Figure 7 shows that the computation of $F_{n,n}$ for $n \leq 21$ can be done with 16-bit integers on machines with 256 GB of RAM. Larger lattices require 384 GB ($n = 22$) or 1 TB ($n = 23$).

There is an alternative approach to determine $F_{n,m}$ with less memory. Instead of enumerating the coefficients $A_{n,m}(k)$, one can compute $F_{n,m}(z)$ for $nm$ integer arguments $z$ (again modulo a sufficient number of primes) and then use Lagrange interpolation to recover $F_{n,m}$. This approach requires only a single $b$-bit integer to be stored with each signature. The prize we pay is that now we need $nm$ computations,
each consisting of several runs for a sufficient number of prime moduli. This was the approach we used to compute $F_{22,22}$ on 30...60 machines with 256 GB each.

Yet another method suggests itself if one is mainly interested in evaluating the spanning probability $R_{n,m}(p)$ for a given value of $p$. Here we need to store only a single real number with each signature, and there is no need for multiple runs with different prime moduli. Quadruple precision floating point variables with 128 bit (\_float128 in gcc) allow us to evaluate $R_{n,m}(p)$ with more than 30 decimals precision. We used this approach to compute $R_{n,n}(p)$ for $n = 23, 24$, see Table 4.

Figure 8 shows the time to evaluate $R_{n,n}(p)$ for given value $p$ and with 30 decimals accuracy, measured on a laptop. The time complexity corresponds to the expected
complexity $O(n^4 \tilde{S}_{n,c^*})$. The $n^2$ iterations to build the lattice, combined with the $O(n)$ complexity of the subroutine \texttt{extend} and the $O(\log \tilde{S}_{n,c^*}) = O(n)$ complexity of the associative container \texttt{set} yield a scaling factor $n^4$. The number of signatures $	ilde{S}_{n,c^*} \simeq 0.196 \cdot 2.61^n$ then provide the exponential part of the time complexity.

Despite the exponential complexity, the algorithm is surprisingly fast for small values of $n$, even on a laptop. Systems up to $12 \times 12$ can be computed in less than one second, $16 \times 16$ takes 90 seconds. The largest system that fits in the 16 GByte memory of the author’s laptop is $21 \times 21$. A single evaluation of $R_{21,21}(p)$ on the laptop takes about 9 hours.

3.7. Results

We have computed the generating functions $F_{n,m}(z)$ for $n \leq 22$ and all $m \leq n$. The data is available upon request. Here we will briefly discuss some properties of the coefficients $A_{n,m}(k)$. For simplicity we will confine ourselves to the quadratic case $m = n$.

$A_{nn}(k)$ has a maximum $A_{\text{max}}$ for $k = k_{\text{max}}(n)$. Our data reveals that

\[
k_{\text{max}}(n) = \begin{cases} 
\frac{1}{2}n(n + 1) & (n \leq 4) \\
\frac{1}{2}n(n + 1) - 1 & (4 < n \leq 22)
\end{cases}
\]

(13)

It is tempting to conjecture that $k_{\text{max}}(n) = \frac{1}{2}n(n + 1) - O(\log n)$.

$A_{\text{max}}$ itself scales like $\simeq 2^n$ (Figure 6). This number determines how many prime moduli of fixed width we need to compute $A_{n,n}(k)$ via the Chinese Remainder Theorem.

The width of the maximum at $k_{\text{max}}$ scales like $n^{-1}$, as can be seen in the “data collapse” when plotting $\frac{A_{n,n}(k)}{A_{\text{max}}}$ versus $n(k - k_{\text{max}})$ (Figure 9).
4. Critical Density

The generating functions $F_{n,m}(z)$ can be analyzed in various ways. One can, for example, investigate their properties for arguments outside the “physical” interval $z \geq 0$. This is how the parity property (6) was discovered and then proven for a large, quite general class of lattices [27].

In this paper we use the generating function to compute finite size estimators for the critical density $p_c$ that we can then extrapolate to the infinite lattice.

4.1. Estimators

The properties of various finite-size threshold estimators for two-dimensional percolation have been discussed by Ziff and Newman [34]. Here we will focus on estimators $p_{\text{med}}(n)$ and $p_{\text{cell}}(n)$, defined as

$$R_{n,n}(p_{\text{med}}) = \frac{1}{2}$$

and

$$R_{n,n}(p_{\text{cell}}) = R_{n-1,n-1}(p_{\text{cell}})$$

The values for these estimators up to $n_{\text{max}} = 24$ are shown in Table 4.

Both estimators are supposed to converge with the same leading exponent,

$$p_{\text{med}}(n) - p_c \simeq c_{\text{med}} n^{-1 - 1/\nu} \quad \text{and} \quad p_{\text{cell}}(n) - p_c \simeq c_{\text{cell}} n^{-1 - 1/\nu}$$

with $\nu = 4/3$. This scaling can clearly be seen in the data (Figure 10).

Any convex combination of finite size estimators is another finite size estimator. We can choose $\lambda$ in $\lambda p_{\text{med}}(n) + (1 - \lambda)p_{\text{cell}}(n)$ such that the leading order terms of $p_{\text{med}}(n)$ and $p_{\text{cell}}(n)$ cancel:

$$\lambda = \left(1 - \frac{c_{\text{med}}}{c_{\text{cell}}} \right)^{-1}.$$
The precise values of \( c_{\text{med}} \) and \( c_{\text{cell}} \) are not known, but their ratio follows from scaling arguments [34]:

\[
\frac{c_{\text{med}}}{c_{\text{cell}}} = -\frac{1}{\nu} = -\frac{3}{4}.
\]

Hence we expect that the estimator

\[
p_\star(n) = \frac{4}{7} p_{\text{med}}(n) + \frac{3}{7} p_{\text{cell}}(n)
\]

converges faster than both \( p_{\text{med}}(n) \) and \( p_{\text{cell}}(n) \). This is in fact the case, as can be seen in Figure 10.

Figure 10 also depicts the estimator \( p_{\text{pol}}(n) \), which is the root of a graph polynomial defined for percolation on the \( n \times n \) torus [35, 36, 37]. Empirically, it converges like \( \sim n^{-4} \) in leading order. Jacobsen [24] computed this estimator for \( n \leq 21 \) and extrapolated the results to obtain

\[
p_c = 0.592746050792(2).
\]

This is the most precise value up to today. We have used it as reference \( p_c \) in Figure 10 and we will continue to use it as reference in the rest of the paper.
4.2. Exponents

Extrapolation methods are commonly based on the assumption of a power law scaling

\[ p(n) = p_c + \sum_{k=1} A_k n^{\Delta_k} \]  

(20)

with exponents \(0 > \Delta_1 > \Delta_2 > \ldots\). These exponents can be determined by the following procedure.

We start by considering the truncated form \( p(n) = p_c + A_1 n^{\Delta_1} \). From each triple of consecutive data points \( p(n) \), \( p(n-1) \) and \( p(n-2) \) we can compute three unknowns \( p_c \), \( A_1 \) and \( \Delta_1 \). This gets us sequences \( A_1(n) \) and \( \Delta_1(n) \) that we can use to compute \( \Delta_1 = \lim_{n \to \infty} \Delta_1(n) \) and \( A_1 = \lim_{n \to \infty} A_1(n) \). Figure 11 shows \( \Delta_1(n) \) for \( p(n) = p_{\text{med}}(n) \) plotted versus \( n^{-1} \) along with a 4th-order polynomial fit. The result \( \Delta_1 = -\frac{7}{4} \) is of course expected. It is stable under variations of the order of the fit and the number of low order data points excluded from the fit.

We then subtract the sequence \( A_1 n^{\Delta_1} \) from \( p(n) \) to eliminate the leading scaling term \( n^{\Delta_1} \). Considering again a truncated form \( p(n) - A_1 n^{\Delta_1} = A_2 n^{\Delta_2} \), we can get sequences \( A_2(n) \) and \( \Delta_2(n) \) from three consecutive terms of this sequence. And again we extrapolate these sequences by fitting a low order polynomial in \( n^{-1} \), thereby obtaining \( \Delta_2 \) and \( A_2 \).

Obviously, this can be iterated to determine the exponents \( \Delta_k \) one by one, although the stability of the procedure deteriorates to some extent with each iteration.

\[ \Delta_1, \Delta_2, \Delta_3, \Delta_4 \]

\[ \Delta_1, \Delta_2, \Delta_3, \Delta_4 \]

\[ \Delta_1, \Delta_2, \Delta_3, \Delta_4 \]

\[ \Delta_1, \Delta_2, \Delta_3, \Delta_4 \]

\[ \Delta_1, \Delta_2, \Delta_3, \Delta_4 \]
Figure [11] shows the extrapolations for $\Delta_2$, $\Delta_3$ and $\Delta_4$ for $p_{med}$ with results

$$\Delta_2 = -2.7516 \approx -\frac{11}{4} \quad \Delta_3 = -3.7553 \approx -\frac{15}{4} \quad \Delta_4 = -4.7590 \approx -\frac{19}{4}$$  \hspace{1cm} (21)

The results for $p_{cell}$ are similar. These results provide compelling evidence that

$$\Delta_k = -\frac{4k + 3}{4} \quad (p_{med}, p_{cell}).$$  \hspace{1cm} (22)

These are the exponents that we will use in our extrapolations.

Applying the same procedure, Jacobsen [24] concluded that the scaling exponents

for $p_{pol}$ are

$$\Delta_k = -2k - 2 \quad (p_{pol}).$$  \hspace{1cm} (23)

For $p_*(n)$, we could not reliably compute exponents $\Delta_k$. It can already be sensed from the curvature in the logarithmic plot (Figure 10), that $p_*(n)$ is not determined by a set of well separated exponents. This is the reason why we will not use $p_*(n)$ for extrapolation in this contribution, despite its fast convergence.

4.3. Extrapolation

A study of extrapolation methods in statistical physics [38] demonstrated that Bulirsch-Stoer (BST) extrapolation [39] is a reliable, fast converging method. It is based on rational interpolation. For a given sequence of exponents $0 < w_1 < w_2 < \ldots$ and $n$ data points $p(1), \ldots, p(n)$, one can compute coefficients $a_0, \ldots, a_N$ and $b_1, \ldots, b_M$ with $N + M + 1 = n$ such that the function

$$Q_{N,M}(h) = \frac{a_0 + a_1 h^{w_1} + a_2 h^{w_2} + \cdots + a_N h^{w_N}}{1 + b_1 h^{w_1} + b_2 h^{w_2} + \cdots + b_M h^{w_M}}$$  \hspace{1cm} (24)

interpolates the data: $p(k) = Q_{N,M}(1/k)$ for $k = 1, \ldots, n$. The value of $a_0$ yields the desired extrapolation $p(k \to \infty)$.

The best results are usually obtained with “balanced” rationals $N = \lceil (n - 1)/2 \rceil$ and $M = \lfloor (n - 1)/2 \rfloor$. This is what we will use for our extrapolations.

The BST method discussed in [38] uses exponents $w_k = k\omega$, where the parameter $\omega$ is chosen to match the leading order of the finite size corrections. Here we choose $w_k = -\Delta_k = -\frac{4k + 3}{4}$.

Having fixed the exponents and the degrees of nominator and denominator, we still have a free parameter: the number of data points to be included in the extrapolation. Since we are interested in the asymptotics $n \to \infty$ we will of course use the data of the large systems, but we may vary the size $n_0$ of the smallest system to be included. We expect that the extrapolation gets more accurate with increasing $n_0$, but only up to a certain point after which the quality deteriorates because we have to few data points left. This intuition is confirmed by the data (Figure 12).

The accuracy of an extrapolation based on system sizes $n_0, \ldots, n_{\text{max}}$ is gauged by the difference between extrapolations of data points for $n_0 + 1, \ldots, n_{\text{max}}$ and $n_0, \ldots, n_{\text{max}} - 1$. These are the errorbars shown in Figure 12.

It seems reasonable to focus on the regime of $n_0$ where the results form a plateau with small errorbars, and to take the mean of these values as the final estimate of $p_c$. Using the values for $n_0 = 14, 15$ ($p_{med}$) and $n_0 = 15, 16$ ($p_{cell}$) (see Table 5) we get

$$p_c = 0.592746050786(3) \quad (p_{med}),$$  \hspace{1cm} (25a)

$$p_c = 0.59274605060(8) \quad (p_{cell}).$$  \hspace{1cm} (25b)
The first value deviates from the reference value \( p_c \) by 2 errorbars, the second by 2.5 errorbars.

The reference value \( p_c \) was obtained with different extrapolation approach. To see how consistent our method is, we applied it also to the estimator \( p_{\text{pol}}(n) \) from Jacobsen \[24, Table. 2\]. Here we used the exponents \[23\] in the BST extrapolation.

Figure 13 shows, that the errorbars are smaller than the errorbars on \( p_{\text{med}} \) and \( p_{\text{cell}} \) (note the scaling of the \( y \)-axis). The estimator \( p_{\text{pol}} \) is also more robust under variation of \( n_0 \). Taking the average of the estimates for \( n_0 = 1, \ldots, 10 \) gives

\[
p_c = 0.592\,746\,050\,792\,0(4) \quad (p_{\text{pol}}).
\]

This value agrees with Jacobsen’s value \[19\] within the errorbars.

We believe that the superiority of the estimator \( p_{\text{pol}} \) is due to the better separation of scales: \( \Delta_{k+1} - \Delta_k = -1 \) for \( p_{\text{med}}, p_{\text{cell}} \) versus \( \Delta_{k+1} - \Delta_k = -2 \) for \( p_{\text{pol}} \).
5. Conclusions

We have introduced a transfer matrix algorithm to compute the generating function $F_{nm}(z)$ for the number of spanning configurations in an $n \times m$ square lattice. Adapting this algorithm to other 2d lattices and other boundary conditions (cylindrical, helical) is straightforward.

The exponential number of signatures imposes a hard limit on system sizes. With the current algorithm, computing $F_{25,25}(z)$ would require a computer with a 4 TB memory and a couple of months CPU time. Evaluating $R_{25,25}(p)$ for a single real argument $p$ requires 512 GB of memory and would take about one month.

We have also seen another example where the exact solution of small systems is sufficient to compute estimates for the critical density $p_c$ with an accuracy that no Monte-Carlo method could ever achieve. Or, as Bob Ziff put it, “Monte-Carlo is dead” [40] when it comes to computing $p_c$ for 2d systems.

This approach can be advanced. Not so much by solving larger systems, but by a better understanding of finite size estimators and their scaling.

Acknowledgements

This paper benefitted a lot from discussions with Bob Ziff. Like almost all of my previous papers on percolation! Whenever I had a question or a crazy idea on percolation, I could ask Bob. And I always got a quick and satisfying answer.

It is a great pleasure to thank Bob for his support and his friendship.

Appendix A. The Number of Signatures

To compute the number $S_n$ of signatures in fully completed rows, we use the connection of signatures and balanced strings of parentheses.
The number of balanced strings of parentheses length $2a$ is given by the Catalan number $C_a$ \[\text{Theorem 7.3}],
\begin{equation}
C_a = \frac{1}{a+1} \left( \frac{2a}{a} \right) = \frac{(2a)!}{(a+1)!a!} = \prod_{k=2}^{a} \frac{a+k}{k}.
\end{equation}

A completely balanced string of parentheses contains the same number $a$ of opening and closing parentheses. We will see that it is useful to consider also strings of parentheses that consist of $a$ opening parentheses and $b \leq a$ closing parentheses such that reading from left to right, every closing parenthesis matches an opening parenthesis. The number of these partially balanced strings of parentheses reads \[\text{[42]}\]
\begin{equation}
C_{a,b} = \frac{a-b+1}{a+1} \left( \frac{a+b}{b} \right) \quad (a \geq b \geq 0).
\end{equation}

The numbers $C_{a,b}$ are known as Catalan’s triangle, and the Catalan numbers are the diagonal $C_a = C_{a,a}$, of this triangle.

**Appendix A.1. From Catalan to Motzkin numbers**

In a signature, the parentheses are not independent. Occupied sites that are adjacent in a row belong to the same cluster. We deal with this local connectivity by considering runs of adjacent occupied sites as single entities that we may call h(orizontal)-clusters. A row of width $n$ contains at most $\lceil n/2 \rceil$ h-clusters. Different h-clusters may be connected through previous rows, and again we have a unique left-right order and the topological constraints that allow us to identify h-clusters with strings of balanced parentheses expression, using the set of symbols (,,),,(, and ). Any legal signature with $r$ h-clusters corresponds to a balanced string of $2r$ parentheses and vice versa.

We write the occupation pattern of a row as $n$-bit word $s \in \{0,1\}^n$, where 0 represents an empty site and 1 represents an occupied site. Let $r(s)$ denote the number of h-clusters, i.e. runs of consecutive 1s, in $s$. Then the number of signatures is
\[\sum_{s \in \{0,1\}^n} C_{r(s)}.
\]
This sum can be simplified by computing the number of $n$-bit words with exactly $r$ h-clusters. For that we add a leading zero to the word. This ensures that every h-cluster has a zero to its left. The position of this zero and the position of the rightmost site of the h-cluster specify the h-cluster uniquely. Hence the number of $n$-bit words with $r$ h-clusters is given by $\binom{n+1}{2r}$, providing us with
\begin{equation}
\sum_{r=0}^{\lfloor n/2 \rfloor} \binom{n+1}{2r} C_r \equiv M_{n+1},
\end{equation}
where $M_n$ is known as the $n$th Motzkin number \[\text{[43]} \text{[44].}

**Appendix A.2. The spanning constraint**

Up to this point we have ignored that our signatures have an additional symbol || for occupied sites connected to the top row. In a signature with $r$ h-clusters, any number
Exact Percolation Probability on the Square Lattice

Figure A1. The dashed connection is forbidden: h-clusters can not connect across h-clusters that are connected to the top row.

$w \leq r$ of h-clusters can be connected to the top row:

$$S_n \leq \sum_{r=1}^{\left\lceil \frac{n}{2} \right\rceil} \sum_{w=1}^{r} \binom{n+1}{2r} \binom{r}{w} C_{r-w}.$$  \hspace{1cm} (A.4)

This is only an upper bound because it does not take into account that h-clusters not connected to the top row can not be connected to each other across a top row h-cluster. In the example shown in Figure A1, the number of signatures compatible with the top row h-cluster is not $C_6 = 132$ but $C_3 C_1 C_2 = 10$.

We need to consider the compositions that $w$ top row h-clusters can impose upon the $r-w$ other h-clusters. The composition of an integer $n$ into $k$ parts is defined as an ordered arrangement of $k$ non-negative integers which sum to $n$. The compositions of 4 into 2 parts are

$$(0, 4), (1, 3), (2, 2), (3, 1), (4, 0).$$

A number of $w$ top row h-clusters can divide the $r-w$ other h-clusters into $w+1$ parts at most. This gets us

$$S_n = \sum_{r=1}^{\left\lceil \frac{n}{2} \right\rceil} \binom{n+1}{2r} \sum_{w=1}^{r} \sum_{i_1+\ldots+i_{w+1}=r-w} C_{i_1} \cdots C_{i_w+1}.$$  

Using Catalan’s convolution formula [46] for $1 \leq k \leq n$,

$$\sum_{i_1+\ldots+i_k=n} C_{i_1-1} \cdots C_{i_k-1} = \frac{k}{2n-k} \binom{2n-k}{n} \quad (1 \leq k \leq n), \hspace{1cm} (A.5)$$

we can write this as

$$S_n = \sum_{r=1}^{\left\lceil \frac{n}{2} \right\rceil} \binom{n+1}{2r} \sum_{w=1}^{r} \frac{w+1}{2r-w+1} \binom{2r-w+1}{r+1}.$$  \hspace{1cm} (A.6)

The sum over $w$ can be done using binomial coefficient identities [47, Table 174]. The result reads

$$S_n = \sum_{r=1}^{\left\lceil \frac{n}{2} \right\rceil} \binom{n+1}{2r} \frac{3}{r+1} \binom{2r}{r-1} = \sum_{r=1}^{\left\lceil \frac{n}{2} \right\rceil} \binom{n+1}{2r} C_{r+1,r-1}, \hspace{1cm} (A.7)$$
Exact Percolation Probability on the Square Lattice

where \((A.7)\) follows from \((A.2)\) with \(a = r + 1\) and \(b = r - 1\). This result can be categorized by considering the generalization of \((A.3)\). Analog to Catalan’s triangle, Motzkin’s triangle \([48]\) is defined as

\[
M_{n,k} = \sum_{r=0}^{\left\lfloor \frac{n-k}{2} \right\rfloor} \binom{n}{2r+k} C_{r+k,r}.
\]  

(A.8)

Hence \(S_n = M_{n+1,2}\), the second column in Motzkin’s triangle. This sequence is number \[A005322\] in the on-line encyclopedia of integer sequences.

Appendix A.3. Asymptotics

Although Motzkin’s triangle is a well known object in combinatorics, we could not find an explicit formula for the asymptotic scaling of \(M_{n,2}\). Hence we compute it here, following the procedure outlined in \([49]\).

Our starting point is the the generating function for \(S_n = M_{n+1,2}\) \([48]\)

\[
F(x) = \sum_{n>0} S_n x^n = \frac{(1 - x - \sqrt{(1+x)(1-3x)})^3}{8x^5}.
\]  

(A.9)

The pole that determines the asymptotics of the coefficients is \(x = 1/3\). Expanding \(F\) in terms of \(\sqrt{1-3x}\) gives

\[
F(x) = 9 - 27\sqrt{3}(1-3x)^{\frac{1}{2}} + \frac{279}{2}(1-3x) - \frac{1485\sqrt{3}}{8}(1-3x)^{\frac{3}{2}} + \cdots.
\]

The coefficients of the binomial series that appear in this expansion are

\[
[x^n] (1-3x)^\alpha = (-3)^n \binom{\alpha}{n} = 3^n \binom{n-\alpha-1}{n} = 3^n \frac{(-\alpha)(1-\alpha)\cdots(n-1-\alpha)}{n!}.
\]  

(A.10)

For \(\alpha = 0,1,2,3,\ldots\), the binomial series is finite and does not contribute to the asymptotic scaling. For \(\alpha \neq 0,1,2,3,\ldots\), the binomial coefficient can be written as

\[
\binom{n-\alpha-1}{n} = \frac{\Gamma(n-\alpha)}{\Gamma(-\alpha)\Gamma(n+1)},
\]  

(A.11)

and we can use the asymptotic series for the \(\Gamma\) function

\[
\frac{\Gamma(n-\alpha)}{\Gamma(n+1)} \sim n^{-\alpha} \left( \frac{1}{n} + \frac{\alpha(\alpha+1)}{2n^2} + \frac{\alpha(\alpha+1)(\alpha+2)(3\alpha+1)}{24n^3} + \frac{\alpha^2(\alpha+1)^2(\alpha+2)(\alpha+3)}{48n^4} + O\left(\frac{1}{n^5}\right) \right).
\]  

(A.12)

to obtain

\[
[x^n] F(x) \sim \frac{\sqrt{3} 3^{n+3}}{\sqrt{4\pi n^3}} \left( 1 - \frac{159}{16n} + \frac{36505}{512n^2} + O\left(\frac{1}{n^3}\right) \right).
\]  

(A.13)
Exact Percolation Probability on the Square Lattice

1. Ziff R M 2009 Efficient simulation in percolation lattices Complex Media and Percolation Theory Encyclopedia of Complexity and Systems Science Series ed Sahimi M and Hunt A G (New York: Springer-Verlag)
2. Hoshen J and Kopelman R 1976 Physical Review B 14 3438–3445
3. Leath P L 1976 Physical Review B 14 5046–5055
4. Alexandrowicz Z 1980 Physics Letters A 80 284–286
5. Grassberger P 1992 Journal of Physics A: Mathematical and General 25 5475–5484
6. Ziff R M 1992 Physical Review Letters 69 2670–2673
7. Ziff R M, Cummings P T and Stell G 1984 Journal of Physics A: Mathematical and General 17 3009–3017
8. Mertens S and Moore C 2018 Physical Review E 98 022120
9. Mertens S and Moore C 2018 Journal of Physics A: Mathematical and Theoretical 51 475001
10. Newman M E J and Ziff R M 2000 Physical Review Letters 85 4104–4107
11. Newman M E J and Ziff R M 2001 Physical Review E 64 016706
12. Scullard C R and Jacobsen J L 2020 Physical Review Research 2 012050(R)
13. Dean P 1963 Proceedings of the Cambridge Philosophical Society 59 397–410
14. Sykes M F and Essam J W 1964 Physical Review 133 A310–A315
15. Dean P and Bird N F 1967 Proceedings of the Cambridge Philosophical Society 63 477–479
16. Levinstein M, Shkolvski B, Shur M and Efros A 1975 Zh. Eksp. Teor. Fiz. 69 386–392
17. Derrida B and Stauffer D 1985 Journal de Physique 46 1623–1630
18. Ziff R M and Sapoval B 1986 Journal of Physics A: Mathematical and General 19 L1169–1172
19. de Oliveira P, Nóbrega R and Stauffer D 2003 Brazilian Journal of Physics 33 616–618
20. Lee M 2007 Physical Review E 76 027702
21. Lee M 2008 Physical Review E 78 031131
22. Feng X, Deng Y and Blöte H W J 2008 Physical Review E 78 031136
23. Yang Y, Zhou S and Li Y 2013 Entertainment Computing 4 105–113
24. Jacobsen J L 2015 Journal of Physics A: Mathematical and Theoretical 48 454003
25. Klamser J U 2015 Exakte Enumeration von Perkolationssystemen Master’s thesis Otto-von-Guericke Universität Magdeburg
26. Yaqubi D, Ghouchan M F D and Zoeram H G 2016 Lattice paths inside a table I arXiv.org:1612.08697
27. Mertens S and Moore C 2019 Physical Review Letters 123 230605
28. Moore C and Mertens S 2011 The Nature of Computation (Oxford University Press) [www.nature-of-computation.org]
29. Kramers H A and Wannier G H 1941 Physical Review 60 252–262
30. Onsager L 1944 Physical Review 65 117–149
31. Derrida B and Vannimenus J 1980 Journale de Physique Lettres 41 L473–L476
32. Cormen T H, Leiserson C E and Rivest R L 1990 Introduction to Algorithms The MIT Electrical Engineering and Computer Science Series (Cambridge, Massachusetts: The MIT Press)
33. Knuth D E 1981 The Art of Computer Programming vol 2: Seminumerical Algorithms (Reading, Massachusetts: Addison-Wesley Publishing Company)
34. Ziff R M and Newman M E J 2002 Physical Review E 66 016129
35. Jacobsen J L and Scullard C R 2013 Journal of Physics A: Mathematical and Theoretical 46 075001
36. Jacobsen J L 2014 Journal of Physics A: Mathematical and Theoretical 47 135001
37. Mertens S and Ziff R M 2014 Physical Review E 90 062152
38. Henkel M and Schütz G 1988 Journal of Physics A: Mathematical and General 21 2617–2633
39. Bulirsch R and Stoer J 1964 Numerische Mathematik 6 413–427
40. Ziff R M 2021 Percolation on non-planar lattices Talk at the 34th M. Smoluchowski Symposium on Statistical Physics
41. Roman S 2015 An Introduction to Catalan Numbers Compact Textbooks in Mathematics (London: Birkhäuser)
42. Bailey D 1996 Mathematics Magazine 69 128–131
43. Motzkin T 1948 Bulletin of the American Mathematical Society 54 352–360
44. Oste R and der Jeugt J V 2015 The Electronic Journal of Combinatorics 22 P2.8
45. Heubach S and Mansour T 2009 Combinatorics of compositions and words. (Boca Raton, FL: CRC Press)
46. Regev A 2012 Integers 12 A29
47. Graham R L, Knuth D E and Patashnik O 1994 Concrete mathematics: a foundation for computer science 2nd ed (Addison-Wesley Publishing Company)
48. Donaghey R and Shapiro L W 1977 Journal of Combinatorial Theory (A) 23 291–301
[49] Flajolet P and Sedgewick R 2009 Analytic Combinatorics (Cambridge University Press)