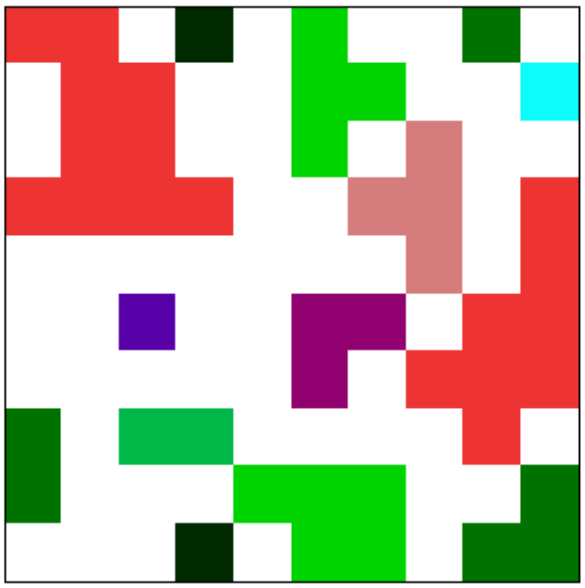
Project in Computational Physics

PERCOLATION

Jan Hasenbusch and Matthias Wilhelm



17/03/2011

Winter semester 2010/11

Contents

1 Introduction

The basic problem of percolation is very simple. On a large lattice of specific type each site is independently either occupied, with probability p, or empty with probability $1 - p$. The occupied sites form clusters which populate the lattice. What is the minimum probability p_c such that a large cluster exists which spans the whole lattice?

This problem was first investigated in the context of vulcanisation of rubber by Paul Flory, who published the first percolation theory in 1941 [9]. There, a slightly different version of percolation appears: assuming a bond between two sites exists with probability p – what is the critical probability for a spanning cluster to exist, now? This version is called bond percolation, whereas the former is called site percolation. Subsequently combinations of both were examined, as well.

Since 1957 computer simulations play a crucial role in percolation theory [9]. Though many interesting quantities are known exactly today, simulations in many cases provided the necessary intuition for determining them and most interesting quantities are still only accessible through simulations altogether.

Percolation models can be used in a huge variety of different systems. Besides actual percolation of fluids through rock, percolation theory was applied to describe granular materials, composite materials, polymers and concrete. Further reaching applications include resistor models, forest fires, epidemics, robustness of the Internet, biological evolution and social influence [6].

To this end, a large number of different lattices is employed.

In this project, however, we focus on site percolation on square lattices in d dimensions.

In section 2 we introduce the fundamental quantities of percolation theory and give an introduction to finite size scaling. Section 3 presents different algorithmic approaches. The implementation of our algorithm is described in section 4 and section 5 contains the results we obtained with it. Our conclusion can be found in section 6.

2 Basics of percolation theory

2.1 Basic definitions

In spite of its relative simpleness percolation exhibits a number of interesting properties. One of these is a second order phase transition.

For low occupation probabilities only small, separate clusters exist in the infinite lattice. For large values of p a single infinite cluster exists that spans the whole lattice. This is shown in figure 1. The value of p for which this infinite cluster appears for the first time is called critical probability p_c. It marks the position of the phase transition.

The order parameter that is associated with the phase transition is called $P_\infty(p)$. It gives the probability for an occupied site to belong to the infinite cluster.

Another fundamental observable is $n_l(p)$, the number (per site) of clusters con-

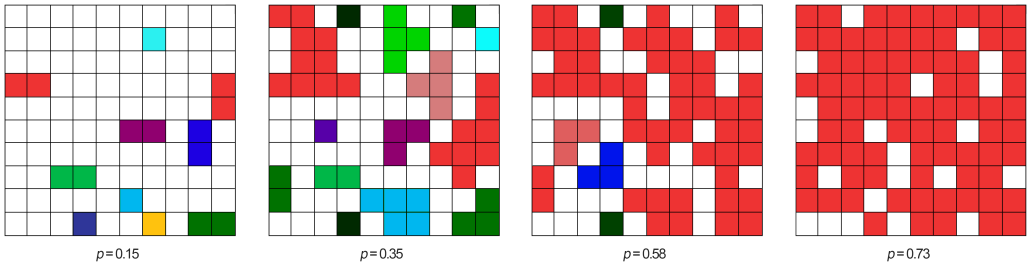


Figure 1: Square lattices for different values of p ($p_c \approx 0.59$)

taining l sites. It is connected to $P_\infty(p)$ by [8]:

$$P_\infty(p) = p - {\sum_l}' l n_l(p) \tag{1}$$

The prime signifies that the infinite cluster is omitted from the sum.

From the number of l-clusters the mean cluster size – or susceptibility – can be calculated:

$$\chi(p) = {\sum_l}' \frac{l^2 n_l(p)}{p} \tag{2}$$

In the literature, its definition varies by an overall factor that is, however, unimportant for its critical behaviour. We work with the definition given in [7].

These quantities can also be derived by formally defining a free energy F as generating function of a ghost field h [8]:

$$F(h) = {\sum_l}' n_l \exp(-hl) \tag{3}$$

Its first h-derivative at $h = 0$, $-\sum_l' l n_l$, is connected to $P_\infty(p)$ and its second one, $\sum_l' l^2 n_l$, to $\chi(p)$. This shows the relation between percolation and thermal physics.

If we define the radius of gyration R_l for a given l-cluster by:

$$R_l^2 = \frac{1}{2} \sum_{i,j=1}^{l} \frac{|\boldsymbol{r}_i - \boldsymbol{r}_j|^2}{l^2} \ , \tag{4}$$

we also arrive at an expression for the correlation length ξ [9]:

$$\xi^2 = \frac{2 \sum_l' R_l^2 l^2 n_l}{\sum_l' l n_l} \tag{5}$$

These quantities exhibit the following critical behaviour near p_c [9]:

$$P_\infty \propto (p - p_c)^\beta \tag{6}$$

$$\chi \propto |p - p_c|^{-\gamma} \tag{7}$$

$$\xi \propto |p - p_c|^{-\nu} \ , \tag{8}$$

which defines the critical exponents β, γ and ν.

Further critical exponents exist with respect to the behaviour of F, n_l and R_l. For a discussion of these and the hyperscaling relations between them we refer the reader to [9].

2.2 Finite size scaling

With a finite computer it is impossible to simulate infinite lattices. Fortunately, nature does not know infinite lattices either. Thus, instead of ruining the theory, the effects of finite size and their understanding give us a further tool to describe observation and make predictions (see [9] for an application to the exploration of oil reservoirs).

In the following, we discuss the effects of finite sizes and sketch the theory of finite size scaling.

In a finite cluster no infinite spanning cluster can exist. To account for this, a finite lattice is said to have percolated if a cluster exist that connects its boundaries (in the case of free boundaries) or wraps once around it (periodic boundary conditions).

Whether the linear dimension L of a finite lattice is relevant for an observable is determined by the correlation length ξ. For $\xi \ll L$ all observables are governed by ξ [9]. For $L \ll \xi$ they are governed by L. Because the correlation length diverges as $p \to p_c$ finite size effects are always relevant in the vicinity of the phase transition.

If an observable X is expected to scale as $|p - p_c|^{-\lambda}$ in the infinite lattice, finite size scaling theory predicts it to obey the general scaling law [9]:

$$X(L, p) = (p - p_c)^{-\lambda} \tilde{X}((p - p_c)L^{1/\nu}) \tag{9}$$

Here, $\tilde{X}$ is a scale-independent function (at least in the asymptotic limit).

In a finite setting the probability $\Pi(p)$ that a given lattice percolates is no longer a step function of p – rising from zero to one at p_c – but varies smoothly. We can define the average concentration p_{av} at which a lattice percolates as [9]:

$$p_{av} = \int p \left(\frac{\mathrm{d}\Pi}{\mathrm{d}p} \right) \mathrm{d}p \tag{10}$$

Applying (9) to $\Pi(p)$ then gives [9]:

$$|p_c - p_{av}| \propto L^{-1/\nu} \tag{11}$$

For practical purposes we use a different formulation of (9) for P_∞ and χ [1, 10]. This yields:

$$P_\infty(L, p) = L^{-\beta/\nu} \tilde{P}_\infty(L^{1/\nu}(p - p_c)/p_c) \tag{12}$$

and

$$\chi(L, p) = L^{\gamma/\nu} \tilde{\chi}(L^{1/\nu}(p - p_c)/p_c) \ . \tag{13}$$

3 Algorithmic solutions

To learn something about percolation one can simply generate configurations for different values of p and analyse these. To do this efficiently has, however, been a challenge which could only be mastered recently.

3.1 Hoshen-Kopelman labelling

An algorithm which was used for many years to investigate percolation configurations is called Hoshen-Kopelman labelling [8].

Perhaps the simplest approach to the classification of clusters is to go through the lattice sequentially and label each seemingly new cluster with consecutive numbers. Figure 2 shows an example of this.

Figure 2: Illustration of Hoshen-Kopelman labelling [8]

The problem of this approach is obvious. Once we come to a site that merges two previously separate clusters, a label conflict exist.

One way to solve this is to relabel all sites with the higher label. However, this necessitates a vast amount of relabelling and is thus very inefficient.

An efficient solution was found by J. Hoshen and R. Kopelman in 1976 [3]. They gave each site label m another index $n(m)$. As long as m is a "good" label, $n(m)$ is m. But once the cluster labelled with m turns out to be a sub-cluster of a cluster labelled with $k < m$, $n(m)$ is set to k. By traversing the resulting tree structure up to the root label the "good" label for each cluster site can be found.

With Hoshen-Kopelman labelling a given configuration can be analysed in time $\mathcal{O}(N)$, where N is the number of sites in the lattice [3]. This is already optimal, as one cannot check the status of N sites in less than N operations [6].

3.2 Microcanonical and canonical ensemble

To study finite size effects and the phase transition the observables in question have to be determined for a large range of p – if not for the whole interval $[0, 1]$ at least for a continuous neighbourhood of p_c. This task can be simplified considerably by switching from the canonical to the microcanonical ensemble.

So far we have only considered a setting where each site is occupied with probability p and empty with probability $1 - p$. The resulting number of occupied sites fluctuates about the mean value pN. In the language of thermal physics this setting is called "canonical" – at least if we replace "energy" with "occupied sites".

It is equally possible to study the microcanonical setting – a lattice in which exactly n sites are occupied. If $Q(p)$ is an observable in the canonical setting we denote its corresponding observable in the microcanonical setting by Q_n.

The desired observable $Q(p)$ can be obtained from Q_n by a convolution with the binomial distribution [5]:

$$Q(p) = \sum_{n=0}^{N} \binom{N}{n} p^n (1-p)^{N-n} Q_n \tag{14}$$

The main advantage of this is that – though there are uncountable many possible values of p – the number of possible values of n is only $N + 1$. Using the microcanonical ensemble it is thus possible to determine $Q(p)$ for all p at once – and this by measuring only $N + 1$ values.

3.3 Newman-Ziff algorithm

In 2000 M. E. J. Newman and R. M. Ziff published an algorithm which calculates observables for all p at once in time $\mathcal{O}(N)$ [5].

Their algorithm is based on the use of the microcanonical ensemble.

The crucial part, however, is that it does not analyse given configurations but analyses configurations while generating them. The core idea can be sketched as follows:

1. Generate an empty lattice.
2. Occupy one randomly chosen previously unoccupied site.
3. Amalgamate clusters that are merged by the newly occupied site.
4. Analyse the changes in the observables due to the newly occupied site.
5. Repeat steps 2 to 4 until all sites are occupied.

By means of a suitable design the changes in the lattice and in the observables due to adding one occupied site can be kept small and managed in time $\mathcal{O}(1)$. In this way, the dependence on N is restricted to the number of steps it takes to occupy the whole lattice and an overall dependence of $\mathcal{O}(N)$ is achieved.

The suitable design mentioned above is the use of a tree structure. Each occupied site is considered a node and a pointer is assigned to it. If a new site is occupied in the lattice, essentially three different situations can occur:

1. None of the neighbours of the new site is occupied. In this case the newly occupied site becomes a cluster of its own and its own root node. Its pointer points to itself.
2. There is exactly one cluster neighbouring the newly occupied site. In this case the newly occupied site becomes a part of this cluster. The pointer of the newly occupied site points to the root node of the neighbouring cluster.
3. There is more than one cluster neighbouring the newly occupied site. These clusters then have to be amalgamated. To this end, one of the root nodes of the neighbouring clusters in picked and the pointers of the other root nodes – as well as that of the newly occupied site – are pointed to it.

Thus, trees grow that store one cluster each – in fact quite similarly to the situation in the Hoshen-Kopelman labelling.

So far, however, the number of nodes that have to be traversed to find the root node grows significantly. To limit this, two refinements are applied:

Firstly, the root node of an amalgamated cluster is not picked randomly from those of the neighbouring clusters but chosen to be that of the biggest cluster. For this purpose, the size of each cluster is stored at its root node and updated upon amalgamation.

Secondly, each time the tree is traversed to find the root node belonging to a site, the path that is taken is compressed. This means that the pointers of all nodes that are passed on the way to the root node are pointing to the root node afterwards.

The resulting algorithm is also called "weighted union find with path compression" – "weighted" and "union" as trees are amalgamated and the smaller one becomes a sub-tree of the bigger one – "find with path compression" as the path to the root node is compressed upon finding it.

4 Implementation

For implementing the Newman-Ziff algorithm, we used the programming language C++.

4.1 Random element

The only place where a random element enters the algorithm is through determining the order in which the sites are occupied.

To this end, a permutation of the addresses of the N sites is needed. It can be obtained by a relatively simple algorithm:

1. Create an array A with N entries. Set $A[i] = i$ for all $i = 0 \ldots N-1$.
2. Set $j = 1$.
3. Create a random number k uniformly distributed between j and $N-1$.
4. Swap $A[j]$ and $A[k]$.
5. Increase j by 1.
6. Repeat steps 3 to 5 until $j = N-1$

For the percolation problem a study of different pseudo-random number generators (PRNGs) was carried out by M. J. Lee [4].

Although we do not expect to reach the precision where differences between research-quality PRNGs matter, we chose to work with the one Lee had identified to be the most reliable. It is the MT19937 algorithm – a version of the Mersenne twister generator developed by Matsumoto and Nishimura.

Concretely, we are using the implementation provided by the GNU Scientific Library (GSL). This library also provides the function `gsl_rng_uniform_int` to create uniformly distributed integer random numbers between 0 and $N-j-1$ from this PRNG. Adding j to these supplies the random numbers needed for the algorithm above.

4.2 Basic programme

We developed the core of our programme according to the description which Newman and Ziff give for their algorithm in [5].

Additionally, these authors published a much more detailed description in [6] – also including the C code of their core programme. This paper was, however, not known to us until one week before the submission date of this report. Hence, our programme was developed independently of it.

We designed our programme completely independent of the dimension d of the lattice. The desired dimension and the linear extension L of the lattice can be chosen

upon initialising the programme – determining the number of sites in the lattice to be $N = L^d$.

The main constituent of the programme is an array of integers of length N which we called `Lattice`. For occupied sites it stores the address of a node of the cluster this site belongs to. For empty sites it contains the marker `N`.

Starting with the address of an occupied site and iterating the command `root = Lattice[root]` enables us to determine the address of the root node of each occupied site.

In a second array of integers of length N – called `Clustersize` – we store the sizes of a cluster at the address of its root node.

After initialising these and several other arrays and variables, the order in which the sites are filled is determined according to the directive described in section 4.1. Subsequently, the main loop fills these site by site:

In the first place, the addresses of the sites neighbouring the newly occupied site are determined and written into an array. This is the only step where the exact geometry of the lattice enters. Here, the periodic boundary conditions are implemented, and only a small change in the executing function would allow us to generate lattices of a completely different type.

From the addresses of the neighbouring sites the addresses of the roots of the neighbouring clusters are determined and written into an array as well.

If there is no neighbouring cluster, the address of the newly occupied site (`newposition`) is stored at `Lattice[newposition]` and `Clustersize[newposition]` is set to `1`.

If there are neighbouring clusters, `Lattice[newposition]` is set to the address of the root of the biggest neighbouring cluster and the size of the biggest neighbouring cluster is increased by one. Furthermore, the size of the biggest neighbouring cluster is increased by the sizes of all other neighbouring clusters, their sizes are set to `0` and the entries of their roots in `Lattice` are set to the address of the root of the biggest neighbouring cluster.

Additionally, all quantities corresponding to observables are updated and the paths of the occupied neighbouring sites to the root node are compressed.

4.3 Computing observables

In the following, we describe how the computation of the different observables was incorporated into the core programme.

In order to determine the step in which the system percolates for the first time, we use the method described in [5]. To each occupied site we assign a vector that points to the root node of its cluster.

Usually the components of the difference vector of these vectors are 0 or ± 1 for neighbouring sites. If, however, the cluster has just percolated, the difference vector points (at least) once around the cluster and (at least) one of its components is considerably larger.

We implemented the vectors by an array of arrays of integers of length (N, d). Every time a path is compressed, these vectors have to be compressed as well.

After each amalgamation we compress the paths of the neighbouring sites and thus update the vectors corresponding to them. We then check the differences

between the vector at the newly occupied site and each occupied neighbouring site for components with an absolute value larger then one.

An illustration of this is shown in figure 3.

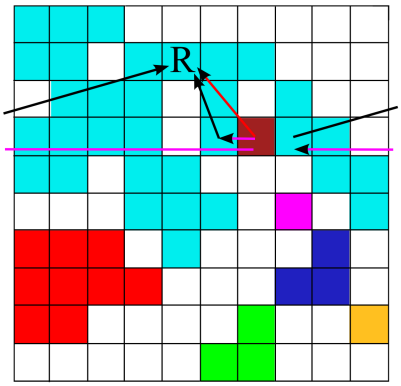


Figure 3: Detecting percolation

Although we determine the step in which the system percolates in each direction separately – and could thus calculate wrapping probabilities as done in [5], we restrict ourselves to calculating p_{av}. Taking the minimum of the steps in which percolation appeared in each direction and dividing it by N gives us an estimate of it.

In order to obtain a reliable value we average p_{av} and all other observable over k runs and output the result. To estimate its error we do this g times.

The final estimate for p_{av} is then given by the mean of the g values and its error can be estimated by the standard deviation of the g values divided by $\sqrt{g}$.

In order to determine $P_{\infty,n}$ we keep track of the size of the biggest cluster in each step. If the system has not yet percolated, $P_{\infty,n}$ is zero. If it has percolated, $P_{\infty,n}$ is given by the size of the biggest cluster divided by n. Although you can think of cases where the spanning cluster is not the biggest one, these cases are extremely improbable. So, statistically, the assumption we made is fully justified.

To determine χ_n we keep track of the sum of the squared sizes of all clusters. For this purpose we use a `long long integer`. χ_n is given by this sum – if the system has percolated minus the size of the biggest cluster – divided by n.

Furthermore, we wrote an extension of our programme to determine $n_{l,n}$. Implemented as N arrays of length $N+1$, the memory necessary for these is enormous even for small values of N. But, fortunately, most of the entries are zero. This allowed us to make effective use of the dynamical data structure of `maps`. At each step we copy the `map` of the previously step, and increase and decrease the sizes of the counters according to the amalgamations.

4.4 Performance

However, most of the time we did not use the extension for the calculation of $n_{l,n}$. The reason for this is that you cannot seriously expect to be able to calculate $\mathcal{O}(N)$ variables in time $\mathcal{O}(N)$.

Figure 4 shows a logarithmic plot of the time necessary for one run for several different N without the extension.

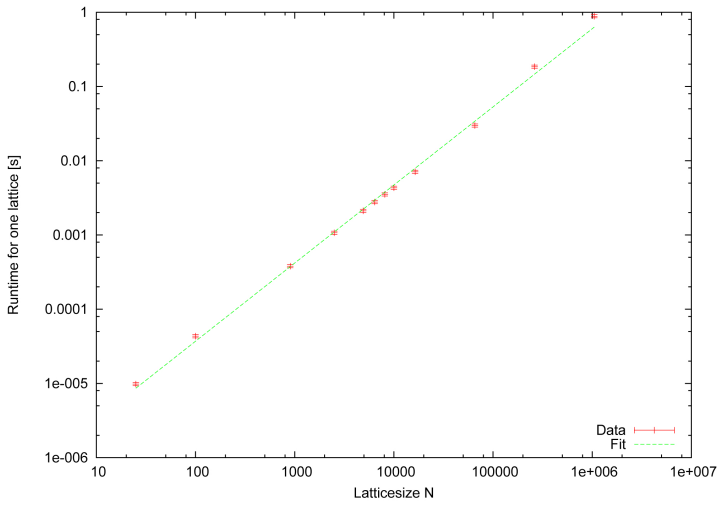


Figure 4: Runtime for different lattice sizes

We fitted the behaviour $\mathcal{O}(N^\alpha)$ and obtained $\alpha = 1.052 \pm 0.017$. This deviates from the desired behaviour. However, according to [6] these deviations can be expected because of inaccuracies in the hardware and are not due to the actual algorithm.

Most of our data was taken with $k = 10,000$ and $g = 10$ resulting in $100,000$ runs per data point.

Our algorithm uses the data type `integer`, which contains numbers up to $2^{31} - 1$ in the implementation of the GNU Compiler Collection (GCC). This limits N to $2,147,483,647$.

However, the real limit is given by the computation time that is available to us. With a reasonably new computer the computation of one run of a $1,024 \times 1,024$ lattice took 0.89 s – which adds up to more than a day for the desired statistics.

In [5] Newman and Ziff used lattices with N only up to 65,536 but 3×10^8 runs arguing that a good statistic is much more important for high precision results than lattice size.

A measurement of the runtime with the extension yielded $\alpha = 1.87 \pm 0.04$.

4.5 Convolution

A convolution with the binomial distribution has to be applied to obtain $P_\infty(p)$ and $\chi(p)$ from $P_{\infty,n}$ and χ_n.

We implemented the binomial distribution taking care to multiply the occurring large and small numbers alternately in order to avoid overflows. However, the computation time needed for the resulting convolutions grew rapidly with growing N.

Therefore, we approximated the binomial distribution by an exponential distribution for $N > 10,000$.

Using the same strategy as for p_{av} we applied the convolutions to the g different outputs for $P_{\infty,n}$ and χ_n separately and determined $P_\infty(p)$, $\chi(p)$ and their errors from the mean values and standard deviations of the results.

To estimate the error due to this approximation we convoluted $P_{\infty,n}$ with both distributions for a lattice with $N = 8,000$. The maximal deviation was 3×10^{-5}, which is considerably less than the maximal statistic error of 4×10^{-4} that occurred at the same value of p. For bigger values of N the deviation becomes even less. Hence, we consider the approximation to be justified in our case.

A more efficient approach for the calculation of the binomial distribution was published by Newman and Ziff in [6]. In this more recent publication they argue that the approximation by an exponential distribution is no longer justified at their desired degree of precision.

5 Results

In total, we took data for $d = 2, 3, 4$ and 6.

From the values of p_{av} we determined the critical probability p_c and the critical exponent ν according to (11). Figure 5 shows the resulting plot for two dimensions.

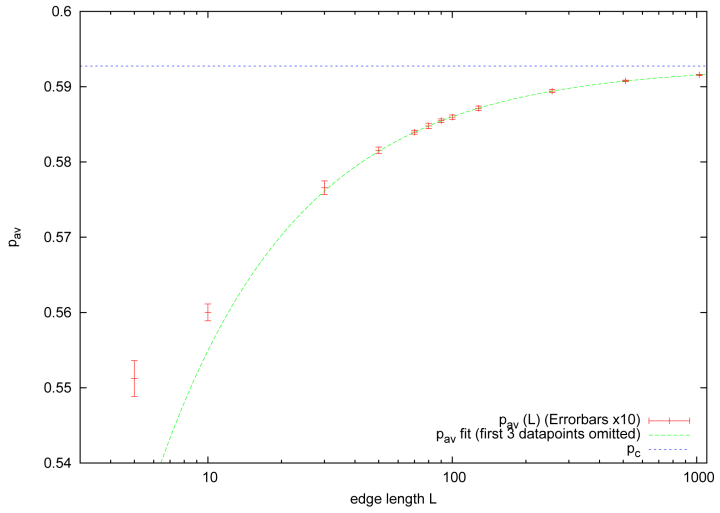


Figure 5: Determination of the critical probability p_c and the critical exponent ν in two dimensions

To determine the critical exponents β and γ we plotted $L^{\beta/\nu} P_\infty(L, p)$ resp. $L^{-\gamma/\nu}\chi(L, p)$ versus $L^{1/\nu}|p - p_c|/p_c$ to find the scaling functions in (12) and (13). We started with the literature values for β and γ and then varied them until the data points for the the different values of L laid on a single curve (compare [1]).

The resulting curves for three resp. two dimensions can be found in figures 6 and 7.

The criterion of overlapping was however not unambiguous and we estimated the errors of the resulting values for β and γ via the range of values for which the data points overlapped reasonably well.

Furthermore, we varied ν to obtain better values for it than could by obtained via p_{av}. This was proposed in [1] and was necessary in three and six dimensions.

Our results for p_c, ν, β and γ, as well as the corresponding literature values, are shown in table 5.

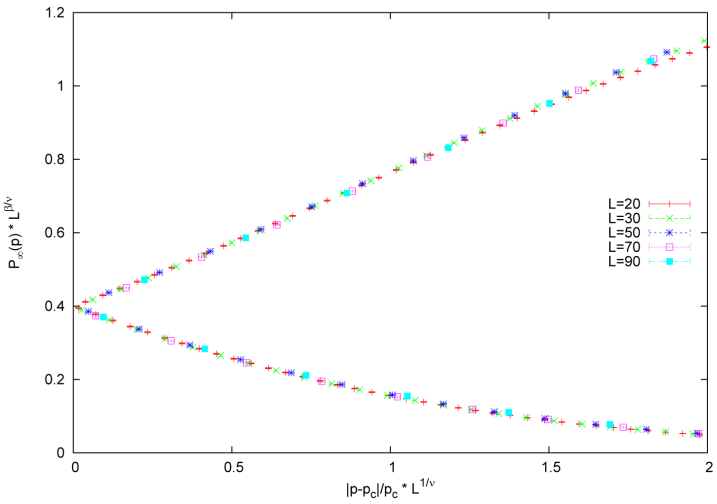


Figure 6: Determination of the critical exponent β for $d = 3$ via finite size scaling of P_∞

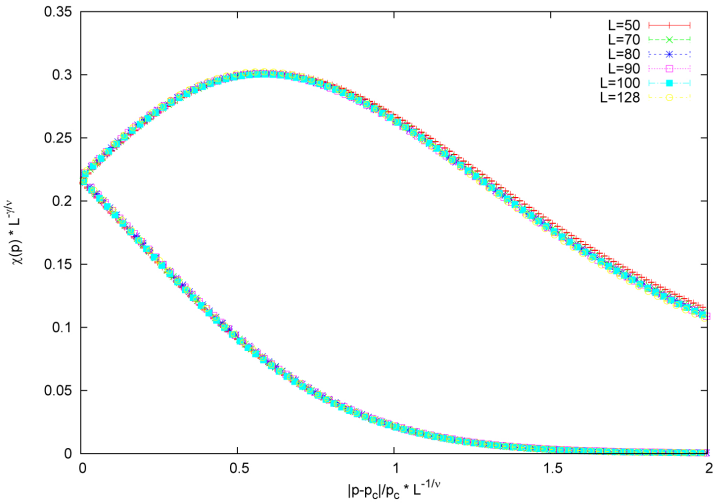


Figure 7: Finite size scaling of the susceptibility χ for $d = 2$

		Our results	Other authors
2D	p_c	0.592737(34)	0.5927621(13)[5]
	ν	1.334(15)	4/3
	β	0.125(2)	$5/36 = 0.1389$
	γ	2.41(2)	$43/18 = 2.389$
3D	p_c	0.31165(6)	0.311608[8]
	ν	0.85(3)*	0.88
	β	0.387(2)	0.41
	γ	1.745(5)	1.769
4D	p_c	0.19693(3)	0.196885[8]
	ν	0.603(14)	0.68
	β	1.19(2)	0.64
	γ	0.64(2)	1.44
6D	p_c	0.1089(2)	0.109018[8]
	ν	0.5(1)*	0.5
	β	1.05(10)	1
	γ	1.35(5)	1

Table 1: Results and literature values. ν-values marked with $\star$ were derived from P_∞ and literature values are taken from [9] if not stated otherwise.

The values we obtained for p_c are very good and match the literature values within one or two standard deviations.

For the critical exponents, however, the quality of our results varies. This is due to the fact that the scaling laws are only valid in the asymptotic limit of $L \to \infty$. For finite L systematic deviations from perfect data overlapping occur, which were clearly visible in some of our plots. These can lead to systematic offsets in the "best-fit" values [1]. With growing d our data becomes increasingly vulnerable to these effects. For example, the largest value for L we could calculate in six dimensions was 10 – this is far less than infinity.

Finally, we measured $n_{l,n}$. Figure 8 contains the result for a 12×12 lattices averaged over 1,000 runs. Each column contains the values for a fixed l and each row those for a fixed n. For better visibility we have multiplied $n_{l,n}$ by Nl/n.

The fractal nature of the system at probabilities close to p_c can be seen very well. There, clusters of all sizes exist, which is exactly the prediction of scale invariance.

6 Conclusion

Albeit founded on simple assumptions percolation exhibits very interesting properties and a lot of interesting applications.

Only recently M. E. J. Newman and R. M. Ziff published an algorithm that is able to calculate observables for all probabilities p at once in time depending linearly on the lattice size.

With a programme based on this algorithm, we investigated site percolation on the square lattice in two, three, four and six dimensions. We were able to determine the critical probability p_c with an accuracy of up to 6×10^{-5}, which we consider to be astonishingly precise for a project of this scope. Furthermore, we determined the

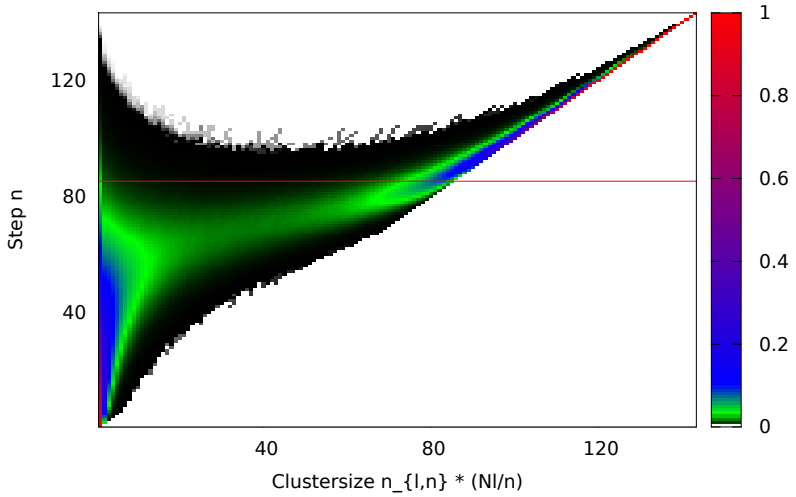


Figure 8: $n_{l,n}$ for a two-dimensional lattice with $L = 12$

critical exponents ν, β and γ. Here, the limits of the computation power available to us became apparent.

However, the available computation time is the only factor limiting the accuracy. And making only small changes it would be possible to apply the programme to the whole range of different lattice types.

7 References

[1] K. Binder and D. W. Heermann. *Monte Carlo Simulation in Statistical Physics, An Introduction*. Springer, Berlin, second corrected edition, 1992.

[2] G. Grimmett. *Percolation*. Number 321 in Grundl. Math. Wissen. Springer-Verlag, New York, second edition, 1999.

[3] J. Hoshen and R. Kopelman. Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm. *Phys. Rev. B*, 14(8):3438–3445, October 1976.

[4] M. J. Lee. Pseudo-random-number generators and the square site percolation threshold. *Phys. Rev. E*, 78(3):031131, September 2008.

[5] M. E. J. Newman and R. M. Ziff. Efficient monte carlo algorithm and high-precision results for percolation. *Phys. Rev. Lett.*, 85(19):4104–4107, November 2000.

[6] M. E. J. Newman and R. M. Ziff. Fast monte carlo algorithm for site or bond percolation. *Phys. Rev. E*, 64(1):016706, June 2001.

[7] K. Ottnad, M. Ronniger, S. Schneider, S. Tölle, and C. Urbach. Projects for Computational Physics, WS 2010/2011.

[8] D. Stauffer. Scaling Properties, Fractals, and the Renormalisation Group Approach to Percolation. *ArXiv e-prints*, April 2007.

[9] D. Stauffer and A. Aharony. *Introduction to percolation theory.* Taylor & Francis, London, 2nd edition, 1992.

[10] A. Sur, J. L. Lebowitz, J. Marro, M. H. Kalos, and S. Kirkpatrick. Monte Carlo studies of percolation phenomena for a simple cubic lattice. *Journal of Statistical Physics*, 15:345–353, November 1976.