RADBOUD UNIVERSITY

Faculty of Science
High Energy Physics Department

A cluster algorithm for simulating 2D Dynamical Triangulations

March 7, 2023

Dion Bremer s1020299
Supervisor: Timothy Budd
Second reader: Harm Schoorlemmer

Abstract

Dynamical Triangulations are a theory that attempts to combine gravity and quantum mechanics. A 2-dimensional toy model of this theory introduces triangulations of the 2-sphere, which can be studied through Markov chain Monte Carlo simulations. In these Markov chains, the triangulation uses an algorithm to form a new triangulation. Previously, this update was performed using a local update move, which results in an autocorrelation time that scales exponentially with the size of the triangulation, with exponent 1.54 ± 0.03. This thesis concerns more global update move, based on the Wolff algorithm for the Ising model, which results in a value of 0.98 ± 0.05. The algorithm also seems useful in the case where an Ising model is attached to a triangulation.

Contents

1 Introduction

Einstein's theory of general relativity is the most accurate description of gravity to date, with many observational successes. General relativity describes how energy and momentum of matter and radiation curve space-time, and how space-time affects the motion of matter and radiation. This notion of space-time makes general relativity a theory of geometry.

However, the theory breaks down when the relevant length scales are close to the Planck length, as quantum fluctuations at this scale are increasingly dominant [12]. Therefore, a quantum theory that explains the behaviour of space-time is necessary.

In order to find such a theory, it is useful to consider the so-called path integral formulation of quantum mechanics. This formulation is equivalent to the well-known Schrödinger formulation of quantum mechanics [3], but takes on a different approach. In this formulation, in order to calculate the probability to find a particle in a certain position, one integrates over all possible paths that the particle could have taken to reach this position [3]. These paths are weighted so that unlikely paths have only a small influence on the outcome, whereas likely paths have a larger influence. In a sense, we integrate over the particle's entire history in order to find its current position. It is useful to note that such paths are continuous objects, but can be discretised into piece-wise linear parts. Shortening these linear parts while increasing the number of parts will return the original path.

A similar approach can be taken to the problem of quantum gravity. Instead of considering a single space-time, we consider the entire history of this space-time geometry, by using a so-called gravitational path-integral [12]. This integral sums over all previous space-time geometries, analogously to the path-integrals. Similar to the path-integrals, we can simplify the integrations by discretising the geometries into regular polygons, which are the piece-wise linear parts in this scenario. These discretised geometries are called *Dynamical Triangulations* [2], *DT* for short, which are the main topic of this thesis. In fact, this thesis concerns only two-dimensional triangulations, which serve as a toy-model to the four-dimensional real-world case.

In order to create a two-dimensional triangulation, imagine cutting a piece of paper into equilateral triangles. Next, glue these triangles together along the edges. Depending on how many triangles meet at each vertex, the resulting geometry can have positive, negative, or no curvature.

While these triangulations provide a model for two-dimensional quantum gravity, they only model empty space. How can one implement some notion of matter into this model? One way to achieve this is to add a spin state to each triangle, which combines the triangulations and the *Ising model*. In the Ising model, a spin state is assigned to a lattice, which then provides a simple model

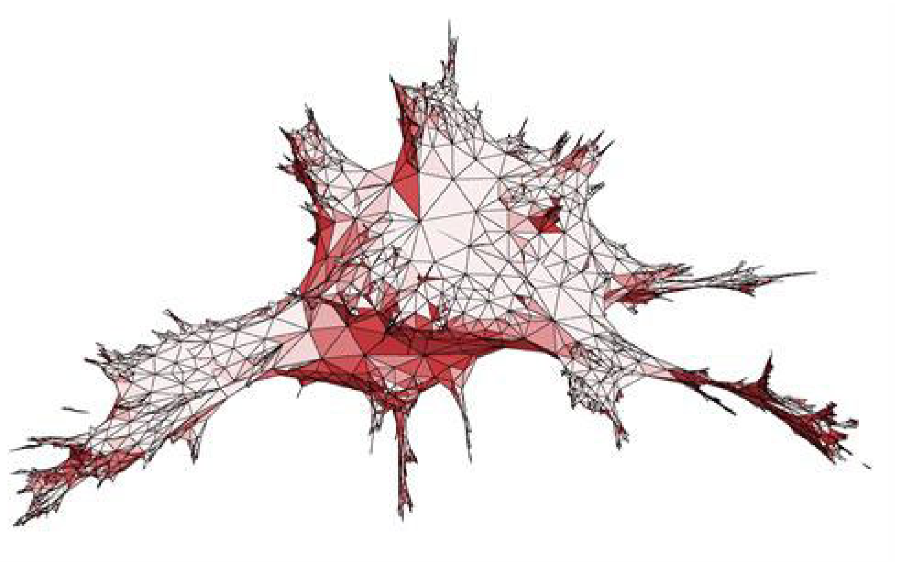


Figure 1: An example of a 2-dimensional triangulation, embedded in $\mathbb{R}^3$. (Figure: Timothy Budd)

for a ferromagnet.

In practice, these models are studied using computer simulations [12], so-called *Monte Carlo simulations*. Using these simulations, we would like to approximate the previously mentioned integral, by averaging over many randomly selected triangulations. This glosses over the fact that the problem of selecting such a random triangulation is not entirely trivial. A solution is to move from triangulation to triangulation, by randomly applying a local update on the first. However, this updated triangulation is not very different from the first, so can this even be called a random selection? In order to make sure that the triangulations are different from each other, it is necessary to make many of these updates instead. This can take a lot of time, especially for very large simulations. Is there a way to improve upon these updates, by applying a non-localised move instead?

This thesis attempts to answer this question, by designing and implementing such a non-localised update algorithm. Firstly, a closer look is taken at DT, as well as the Ising model. Secondly, the simulations themselves are elaborated upon. Thirdly, some observables on triangulations are described, as well as a method of "traversing" a triangulation. Next, the research question is further refined, and a cluster algorithm is described. After this, the measurements on the simulations are presented, and finally, conclusions are drawn, and a discussion on the results and future research is provided.

2 Causal Dynamical Triangulations

As touched upon in the introduction, causal dynamical triangulations (CDT for short) are a proposed candidate theory for the problem of unifying quantum mechanics with gravity [12].

2.1 Gravity

Gravity is best described by the *General theory of relativity.* Central in this theory are Einsteins field equations

$$G_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}, \tag{1}$$

where

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} R g_{\mu\nu}. \tag{2}$$

In these equations, $R_{\mu\nu}$ is the *Ricci tensor*, $g_{\mu\nu}$ is the *metric*, R is the *curvature*, G is the *gravitational constant* and $T_{\mu\nu}$ is the *stress-energy tensor.* Note that in both equations the Einstein summation convention is used. The important part about equation (1) is that the geometry of space-time (on the left) is influenced by the presence of matter, momentum and energy (on the right), and vice-versa. We are often interested in the metric $g_{\mu\nu}$, as this contains the information of the geometry. The theory breaks down at the quantum scale, as quantum fluctuations become too prominent [12].

In other quantum field theories, these fluctuations are *renormalised.* When performing *perturbation theory* on the fields, one encounters infinities, which are then countered by introducing a new term into the equations. If we were to apply the same technique to gravity, and view the metric as a flat space-time with a perturbation $g_{\mu\nu}(x) = \eta_{\mu\nu} + h_{\mu\nu}(x)$, then each order of perturbation theory would require a new counter term [7]. This tells us that perturbation theory may not be the best approach to gravity.

2.2 Triangulations

Therefore, a new quantum-mechanical theory is necessary to describe gravity at the quantum level. The theory that we are concerned with is called *Causal Dynamical Triangulations*, which takes inspiration from *path integrals.*
In the path integral formulation of quantum mechanics, one calculates the values of observables of a system using equation (3) [7], instead of using the Schrödinger equation directly:

$$\int_{x_i}^{x_f} [\mathcal{D}(x)] e^{\frac{i}{\hbar} S[x(t)]}. \tag{3}$$

This equation is an integration over all possible paths from point x_i to x_f. The function $S[x(t)]$ is called the *action*, and is defined as

$$S[x(t)] = \int_{t_i}^{t_f} dt\mathcal{L}(x(t), \dot{x}(t)), \tag{4}$$

where $\mathcal{L}(x(t), \dot{x}(t))$ is the Lagrangian of the system. The term $e^{\frac{i}{\hbar}S[x(t)]}$ in equation (3) can be seen as a weight on each path.

The gravitational analogue to equation (3) [7] is given by

$$\mathcal{Z} = \int [\mathcal{D}g_{\mu\nu}] e^{\frac{i}{\hbar}S[g_{\mu\nu}]}, \tag{5}$$

where

$$S[g_{\mu\nu}] = \frac{c^4}{16\pi G} \int d^4x \sqrt{-g}(R - 2\Lambda) \tag{6}$$

is the *Einstein-Hilbert action.* The integration in equation (5) is performed over all possible geometries, instead of paths. This integral is very difficult to perform formally, so we need to simplify some things. Firstly, assume the space to be Euclidean. The integral now becomes

$$Z = \int [\mathcal{D}g_{ab}] e^{-\frac{1}{\hbar}S_E[g_{ab}]}, \tag{7}$$

where

$$S_E[g_{ab}] = \frac{c^4}{16\pi G} \int d^4x \sqrt{g}(-R + 2\Lambda). \tag{8}$$

Next, we will be working in only 2 dimensions. The final assumption is that any geometry corresponding to the metric g_{ab} can be approximated by a *triangulation.* A triangulation is a manifold which consists of multiple equilateral triangles glued together at the edges. When we increase the number of triangles, and decrease the size of each triangle, we can approximate these geometries. In this thesis, we will always take the approximated space-time manifold to be topologically equivalent to the 2-sphere. The integral in equation (7) now becomes a sum

$$Z = \sum_T e^{-S_{DT}[T]}, \tag{9}$$

where the action S_{DT} depends on the dimension of the system. The 2-dimensional case, which is the case that we are interested in, the action happens to be $S_{DT} = 0$. Equation (9) may look familiar to people who have studied statistical physics before, as it resembles a partition sum often used in this field. This is indeed how it is also used in dynamical triangulations. The partition sum describes a probability distribution. Due to the trivial value of the action, the partition sum becomes $Z = \sum_T 1$, and the corresponding distribution is the uniform distribution over all triangulations of the 2-sphere of size N [7].

2.3 Planar maps

As elaborated upon before, in DT, the geometry of space-time is modeled by "gluing" together triangles along the edges, see figure (1), creating *planar maps*.

A planar map is a connected graph where all vertices are disjoint, and the edges only intersect at the vertices [9]. In practice, these triangulations are studied using computer simulations. Therefore, we need to define a representation of triangulations that we can use in software. To this end, we use *combinatorical descriptions* to model our triangulations [8]. In these descriptions, we use half-edges in order to build triangles. Each half-edge contains the information for the next half-edge, as well as the adjacent half-edge. This means that when we start at a random half-edge and move to the next three times, we end up at the same half-edge. When we move to the adjacent edge, we end up in a new triangle. If we now label all the half-edges, we can define the next and adjacent operators to be permutations that move a label to the next or adjacent label respectively [8], hence the name combinatorical description.

These planar maps make for good candidates for implementation in software, as graphs are well-understood objects in the field of computing science.

2.4 Ising model

Using just triangulations, we can model "empty" spacetime: there is no matter present yet. In order to add matter to the model, we can introduce spin states to the faces of the triangulation, in order to create an Ising model. In the Ising model, there are only two spin states: spin up and spin down. This creates a statistical system, with a corresponding partition sum

$$Z_{Ising} = \sum_{S} e^{\beta J \sum_{(i,j)} \sigma_i \sigma_j}, \tag{10}$$

where the first summation is carried out over all possible spin configurations S, and the second sum is carried out over adjacent spin pairs (i, j). In this equation, σ represents a spin state, which has value 1 for spin up and -1 for spin down. The value of β depends on the temperature $\beta = \frac{1}{k_b T}$ as usual, and J is the coupling constant, which determines whether the spins prefer to align themselves or anti-align.

When attaching the Ising model to a triangulation, the partition sum of the entire system changes to

$$Z_{T+Ising} = \sum_{T} Z_{Ising}(T), \tag{11}$$

where $Z_{Ising}(T)$ means the Ising model partition sum on triangulation T. It is important to note that this sum depends on the triangulation, as the adjacent spin pairs depend on the triangulation.

3 Monte Carlo simulations

Monte Carlo simulations are the central topic of this thesis. They are useful when studying causal dynamical triangulations, but also in many other situations that involve random numbers [7]. The goal of a Monte Carlo simulation is to randomly sample from some set of outcomes, which in our case are random triangulations. This can then be used to approximate the aforementioned gravitational path-integral, see equation (9).

3.1 Markov chains

In order to perform this sampling, we shall use a *Markov chain.* A Markov chain is a sequence of states, in our case triangulations, where the $i+1$-th state follows from performing a random move on the i-th state [7] [1]. It is also possible that a certain move is rejected, since it may construct an impossible situation. In this case, the $i+1$-th state is the same as the i-th state. Markov chains are useful in situations where it is difficult to directly sample states from a state space, such as with triangulations. The starting state of our simulations will be a triangulated 2-sphere.

It is clear that there must be some rules for the moves to ensure that the simulation samples states from the desired distribution. One of these rules is that the move from state i to $i+1$ should satisfy *detailed balance* [7]. This is the requirement that the desired probability of finding the system in state x, and transitioning to state y must be the same as finding the system in state y transitioning to state x,

$$P(x)P(x \to y) = P(y)P(y \to x), \tag{12}$$

where $P(x)$ is the desired probability of finding the system in state x and $P(x \to y)$ is the probability of the system transitioning from state x to state y.

Another important requirement is called *ergodicity.* This requirement states that every state must be reachable from every other state. This requirement ensures that all interesting physical states can actually be reached, and therefore contribute to the simulation.

When both requirements of detailed balance and ergodicity are satisfied, we can be sure that the Markov chain converges to the desired probability distribution.

3.2 Edge flip

As described before, in order to move from the i-th triangulation to the $i+1$-th triangulation, we require some sort of *update move.* The easiest way of performing such a move is the *edge flip* move. By performing this move, an edge

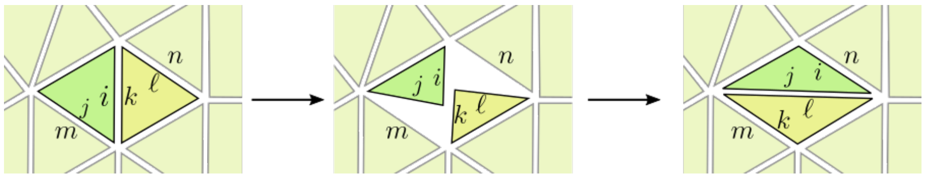


Figure 2: The edge flip move on the edge with half-edges i and k. Since a flip only affects a single edge, it is called a local update to the triangulation. Since a triangulation is invariant under renaming of the labels, the labels themselves are not important. Image from [7]

connecting two vertices is flipped, and now connects two other vertices instead. This move is never rejected, as the resulting state will always be a viable triangulation. By choosing the edge uniformly at random, this move satisfies detailed balance, and is therefore a viable move, see figure (2)

3.3 Cluster move

As one may suspect, the edge flip move mentioned in the previous section does not have much influence on the triangulation as a whole, especially on large triangulations. Here we can draw an analogy to the Ising model. Consider a move that updates a single spin in a large Ising model. This also does not have a big influence on the overall magnetization, and it may take many updates before anything significant happens. Because of this, people often use *cluster moves* to simulate the Ising model. In a cluster move, several spins are updated at once, for a more "global" effect compared to updating a single spin. Particularly, the *Wolff* algorithm is useful in these simulations.

3.4 Wolff algorithm

The Wolff algorithm is an algorithm that is often used in simulations of the Ising model [1]. It uses a cluster to update several spins at once. The algorithm works as follows [1]:

1. Create an empty cluster, as well as an empty stack.
2. Select a random spin, and add this to the cluster, as well as the stack.
3. Pop a spin off the stack, and add all parallel nearest neighbours of this spin to both the cluster and the stack with some probability $0 < p < 1$.
4. Repeat step (3) until the stack is empty.
5. Flip all the spins in the cluster.

The Wolff algorithm is rejection-free, meaning the cluster is always flipped. The Wolff algorithm generates statistically independent configurations quicker than when using single spin flips, making it a useful algorithm in Monte Carlo simulations.

4 Peeling exploration

If we wish to translate the Wolff algorithm to dynamical triangulations, it will be helpful to know how to traverse a planar map. The method that we will be using is called a *peeling exploration* [9]. A peeling exploration is an exploration of a planar map, guided by an algorithm. In a peeling exploration of a triangulation, we keep track of triangles that we have already explored, and triangles that we have not yet explored, called the *holes*.

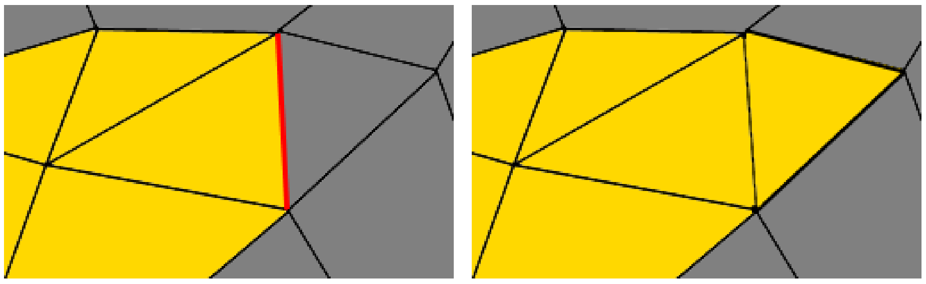
To start the exploration, we should select a triangle to start with. This triangle is called the *root*, and all other triangles are holes. The *boundary* is the set of edges that are between the explored region and the holes [9], so in this case, the boundary consists of the three edges of the root triangle.

The goal of the exploration is now to explore the holes, by repeatedly expanding the boundary. To do so, we define an operation called *peeling*, which takes as arguments the explored region E, an edge e on the boundary, and the full planar map M. Now, $Peel(E, e, M)$ is a new map, that is obtained by gluing the triangle that lies in the hole adjacent to e, onto E.

Formally, a peeling exploration is now the sequence of maps

$$E_0 \to E_1 \to \cdots \to M, \tag{13}$$

where E_0 is only the root face, which eventually becomes the entire map. There is a lot of freedom in this exploration: the exact edge that is peeled each step is determined by some algorithm.



(a) Situation before the peeling operation. The red edge is selected to be peeled.

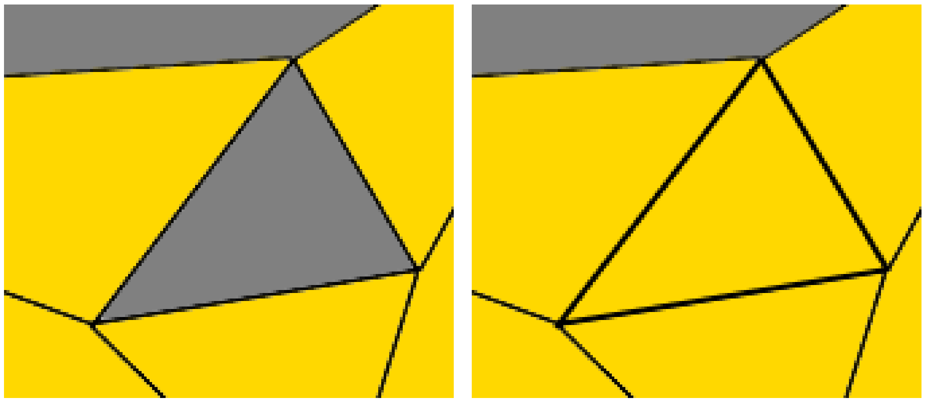
(b) Situation after peeling, where the explored region has expanded.

Figure 3: The peeling operation on a triangulation. The yellow area represents the explored region, while the gray area represents the hole.

With some foresight into the cluster update move of the triangulation, it will prove useful to define a *filling* operation as well. During an exploration, it often happens that the boundary becomes disconnected, or creates more than one cycle. The *Fill* operation resolves this issue, by filling in any internal holes in the explored region. Before the exploration starts, we select a *target* triangle, which is a triangle that is as far away from the root as possible. This triangle will help with deciding which holes to fill. The *Fill* operation takes as arguments the explored region E, as well as the entire map M. Now $\mathit{Fill}(E, M)$ is the region obtained by filling in every hole in the map, except for the hole that contains the target. This means that after a fill operation, the map consists of the explored region and only one hole. A *filled-in peeling exploration* is now the sequence of maps

$$E_0 \to \mathit{Fill}(E_1, M) \to \cdots \to M, \tag{14}$$

which is equal to a regular peeling exploration, but with a *Fill* operation at each *Peel* operation. The use of a target triangle introduces one more issue: what if after a *Peel* the target is explored? Since this would be an issue with regard to the *Fill* operation, we stop the exploration when this happens.


(a) Situation before the fill operation. The boundary is not a single cycle.

(b) Situation after the fill operation. The hole within the boundary is removed.

Figure 4: The fill operation on a triangulation. The yellow area represents the explored region, while the gray area represents the hole.

5 Observables

In order to gauge the "efficiency" of an algorithm, we should keep track of some observables of the triangulations. For formality, denote the space of all possible triangulations as Γ. Then we can define an observable on a triangulation X as a function:

$$f : \Gamma \to \mathbb{R} \tag{15}$$

In order to define some observables, it is useful to view a triangulation as a graph, so that it is possible to define our observables as *graph invariants*. Graph invariants are observables of graphs that are invariant under graph-isomorphisms. Since physical observables may not rely on the representation of the triangulation, this is an important property to have. Note that not all graph invariants define useful observables, for example, the girth (the length of the shortest cycle in a graph) of a triangulation is always 3.
In the remainder of this section, I will use the usual graph definition $G = (V, E)$, where V is the set of vertices and E is the set of edges, to describe triangulations.

5.1 Diameter

The first observable that was studied in this thesis, is the *diameter* of the triangulation. First, we define the *eccentricity* of a vertex $v \in V$ as

$$ecc(v) = \max_{u \in V} d(u, v) \tag{16}$$

where $d(u, v)$ is the shortest distance between vertices u and v. Now, the diameter can be defined as

$$diam = max_{v \in V} ecc(v) \tag{17}$$

The diameter is an important observable, as it encompasses the entire triangulation, and therefore says something about the global structure of the geometry.

In order to calculate the diameter, one could calculate the distances between every pair of vertices in the triangulation, and select the largest value. This is obviously computationally very expensive, as it scales quadratically with the size of the triangulation. A more efficient algorithm to calculate the diameter is the *iterated Fringe Upper Bound* algorithm [10], which calculates these distances in a good order, so that not all of the distances have to be calculated. This could save a lot of computing time.

5.2 Laplacian matrix

Finally, we will consider the *Laplacian matrix*, or, to be more precise, the eigenvalues of this matrix. To construct this matrix, we can first construct the *degree* *matrix* of the triangulation. Let the vertices be labeled by indices i (this is fine, since the eigenvalues of the Laplacian matrix will be graph-invariant). Then the degree matrix is defined as the matrix with the degrees of the vertices on the diagonal:

$$D_{ij} = \begin{cases} deg(u_i), & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} \tag{18}$$

Now define the *adjacency matrix* as

$$A_{ij} = \begin{cases} 1, & \text{if } (u_i, u_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

Now, we can define the Laplacian matrix as the difference between the two:

$$L_{ij} = D_{ij} - A_{ij} \tag{20}$$

This matrix once again contains information about the graph as a whole, and can therefore lead to interesting observables. We will use the smallest eigenvalue of the system as an observable.

In order to calculate these eigenvalues, I used the *Eigen* [11] and *Spectra* [13] C++ libraries. These libraries have functions to construct the Laplacian matrix, and to calculate the first N eigenvalues of this matrix.

5.3 Magnetization

In the case of the Ising model coupled to a triangulation, we can use the *magnetization* as an order parameter, which is calculated as

$$M = \frac{1}{N} \sum_{i=1}^{N} \sigma_i, \tag{21}$$

where N is the number of triangles. More precisely, we will use the absolute value of this observable. It is clear that the magnetization can be measured quite trivially, by simply summing up all spin-values.

5.4 Autocorrelation

These observables will function as *order parameters.* The behaviour of these order parameters will depend on the underlying triangulation, and when an update-move of the triangulation is performed, the parameter may change. However, it should be clear that it may take several moves in order to completely change the value of the observable. Therefore, two measurements of the same observable may take multiple updates to be completely independent of one another. This phenomenon is called *autocorrelation* [7], and associated with this is the *autocorrelation time*, which is what we will use to measure how well our algorithm performs.

We define the autocorrelation explicitly as the correlation between the i-th and the $(i+t)$-th value of the given observable:

$$\rho(t) = Corr(f(X_i), f(X_{i+t})) = \frac{\mathbb{E}\big[\big(f(X_i) - \mathbb{E}[f(X)]\big)\big(f(X_{i+t}) - \mathbb{E}[f(X)]\big)\big]}{Var(f(X))}. \tag{22}$$

However, equation (22) is not easy to work with. Instead, as an estimate, we use the *sample autocorrelation*, defined as:

$$\bar{\rho}(t) = \frac{\bar{\gamma}(t)}{\bar{\gamma}(0)}, \tag{23}$$

where we use the *sample autocovariance*:

$$\bar{\gamma}(t) = \frac{1}{n-t}\sum_{i=1}^{n-t}(f(X_i) - \overline{f(X)}_n)(f(X_{i+t}) - \overline{f(X)}_n). \tag{24}$$

Now one final step is required to calculate the autocorrelation time. To this end, we assume that the sample autocorrelation decays exponentially, so that we can fit an exponential curve to the autocorrelation,

$$\bar{\rho}(t) \propto e^{-t/\tau}, \tag{25}$$

which results in an autocorrelation time τ. Note that this does introduce a systematic error, as the sample autocorrelation needs not decay exponentially. Nevertheless, the autocorrelation time obtained using this method is still useful in most measurements, as an experimenter could take this autocorrelation time into account when performing measurements on a statistical system.

In previous work, it was shown that the autocorrelation time when using edge-flip moves scales as N^α,

$$\tau \propto N^\alpha \tag{26}$$

where $\alpha = 1.54 \pm 0.03$ [5], and N is the number of triangles. We shall use equation (26) in order to determine the value of α in the case of the cluster algorithm. This value will then determine if the cluster algorithm improves upon the edge flips.

5.5 Equilibration

One should be careful when measuring the autocorrelation, however. The problem lies with the fact that the first configuration in the Markov chain is predetermined. Therefore, the first measurements should be ignored, as these are biased by the initial configuration. When is it safe to start measuring? Since we would like to sample from the set of all triangulations, we should wait until the observable is near its expected value. The time it takes until this value is reached, is called the *equilibration time*.

6 Research question

With all the theory in mind, the main research question can be formulated as follows:

Can one develop a cluster algorithm for Dynamical Triangulations and does it improve the simulations' efficiency?

In order to answer the question, it is divided into several sub-questions, some of which are answered in the previous sections.

- What cluster growth algorithms are suitable for the job? Can one leverage the peeling explorations studied in the mathematical literature?
- Observables in Quantum Gravity are generally hard to come by, especially ones that are sensitive to the global aspects of the geometry. What are observables that can be effectively used to gauge autocorrelation times in the simulation?
- How does the cluster algorithm perform compared to a local updating procedure in autocorrelation time (both in units of Markov chain updates and CPU time)?
- Can the cluster algorithm aid in improving the challenging simulation of Dynamical Triangulations coupled to a critical Ising model?

7 Algorithmic design

With all the previous in mind, I designed a cluster algorithm for use in simulations of dynamical triangulations. The Wolff algorithm was used as a basis of the algorithm. It uses the *scale-invariance* that both the triangulations as well as the Ising model exhibit, which means that the same properties of the system appear on all length scales (the system "looks" the same regardless of how far we "zoom in"). The algorithm consists of three steps:

1. Build a cluster of triangles, using a certain probability of adding more faces.
2. Find the correct boundary of the cluster using a target edge.
3. Rotate the cluster in an appropriate manner.

7.1 Building the cluster

As mentioned before, the algorithm is based on the Wolff algorithm, and it generates a cluster in the same way the Wolff algorithm does:

1. Randomly select a face.
2. Add the half-edges of this face to both the stack as well as the boundary.
3. Calculate the edge that is farthest away from the initial face using breadth-first search. This is the target edge.
4. For each edge in the stack, peel the edge with probability p, and remove it from the stack.
5. Continue until the stack is empty, or the target edge is peeled.

The peeling of a half-edge e used in this algorithm is implemented as one of four operations, depending on the adjacent face F. Using the notation $A(e)$ for "the half-edge adjacent to e", $N(A(e))$ for "the half-edge next to $A(e)$" and $P(A(e))$ for "the half-edge next to $N(A(e))$", we get the following situations:

1. If e is not in the boundary, do nothing.
2. If both $N(A(e))$ and $P(A(e))$ are not in the boundary, add them to the boundary and the stack, and remove e from the boundary.
3. If one of $N(A(e))$ and $P(A(e))$ is in the boundary, remove this half-edge from the boundary, and add the other to the boundary and the stack, and remove e from the boundary.
4. If both $N(A(e))$ and $P(A(e))$ are already in the boundary, remove e, $N(A(e))$ and $P(A(e))$ from the boundary.

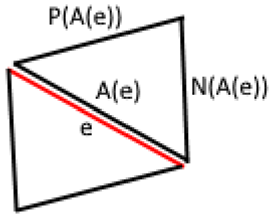


Figure 5: The naming convention for the half-edges. The half-edge e is to be peeled.

These situations all add the face F to the cluster, with the exception of the first situation, by appropriately altering the boundary of the cluster.

This phase will eventually stop when the stack is empty, so that all edges in the boundary have failed to peel, or when the target edge is peeled, or when the entire triangulation has been added to the cluster.

7.2 Finding the boundary

Now that the cluster has formed, there is still a problem to tackle: the cluster most likely has holes in it, so that the boundary is not cyclical. This is why the target edge was introduced at the start of the algorithm. The algorithm uses this edge in the following manner to find the boundary:

1. Run a breadth-first search starting at the target edge to find the boundary element closest to the target.
2. Run a depth-first search from this element, considering only boundary elements, to find the cycle.
3. Retrace the depth-first search tree to find all the elements of the cycle.

The result of this operation will be a cyclical boundary, such that there are no more holes in the cluster.

7.3 Rotating the cluster

At last, a proper cluster has formed, so that it is now possible to perform a cluster operation. First, define the boundary elements as B_i, with $i = 1 \dots N$, with N the size of the boundary. Now define the "outer boundary" as $O_i = A(B_i)$, the half-edges adjacent to the boundary. Next, define an angle of rotation. In this case, the angle is a number in $\{1 \dots N-1\}$. We cut the cluster out of the triangulation, and glue it back together using the angle. Now, $B_{angle+i}$ is glued to O_i, so that the entire boundary is rotated *angle* elements. Obviously, $angle + i$ is calculated modulo N.

This rotation completes the entire cluster move, and after it is performed, the algorithm can be run again.

7.4 Ising model adaption

Due to the fact that the Wolff algorithm on the Ising model adds only parallel spin states to a cluster, it is necessary to make some adjustments to the algorithm when combining triangulations with the Ising model. One change is the obvious change that only faces with parallel spins will be added to the cluster in the build phase. The other change is that all spins in the cluster are flipped before filling in the holes. This is to ensure that all spins that are flipped are indeed parallel (since a hole may contain an anti-parallel spin). The resulting algorithm therefore handles both the flipping of the spins as well as the rotating of the faces with a single cluster.

7.5 Implementation details

The algorithm was implemented in C++, using a template for triangulations provided to me by Timothy Budd. The source code can be found on GitLab [6]. After building the project, the first argument is the desired number of triangles, the second is the number of measurements, the third the number of moves per measurement, the fourth the number of burns before measuring, the fifth the probability of peeling an edge (only used when building clusters), the sixth is the output file and the last argument is the mode, represented by an integer. The mode dictates both the observable that is measured, as well as whether the program uses cluster moves or edge flips. When edge flips are used, the probability value is ignored.

Integer	Mode
0	Diameter in cluster mode
1	Degree sequence in cluster mode
2	Radius in cluster mode
3	Diameter in naive mode
4	Radius in naive mode
5	Degree sequence in naive mode
6	Laplacian in cluster mode
7	Cluster size
8	Magnetization in cluster mode
9	Laplacian in naive mode
10	Magnetization is naive mode

Table 1: Different mode options for the program.

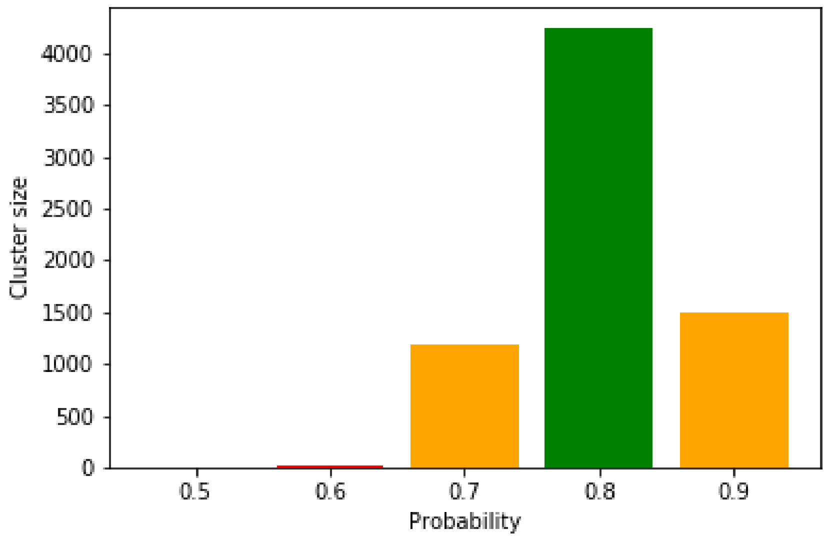


Figure 6: Average cluster size for different probabilities, on a triangulation of 100000 triangles.

8 Measurements

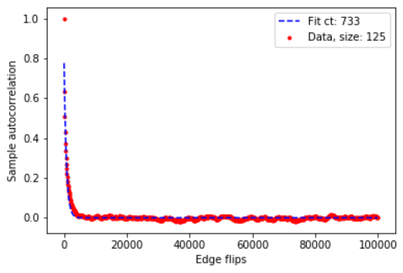
The following section shows the measurements that were performed in order to test the algorithm. I would like to thank the department of High Energy Physics for allowing me to execute the simulations on their cluster computing system.

8.1 Cluster sizes

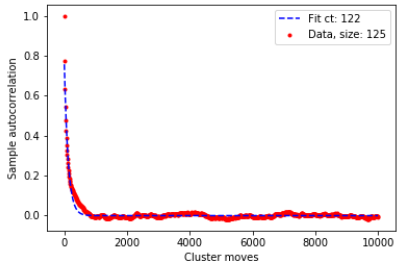
Before running the actual simulations, it is necessary to determine the optimal value for the probability parameter in the algorithm. This parameter will, after all, determine how large the clusters will become. Figure (6) shows the results of a small test, which measured how many triangles each cluster contained. If the cluster contains over half of the triangles in the triangulation, I measured the size of the hole instead, as there is no real difference between rotating the hole and rotating the cluster. What becomes clear from figure (6) is that the right probability lies in the vicinity of 0.8. Interestingly, the field of *percolation theory* suggests that there is a critical threshold in the probability at $p \approx 0.81699$ [4]. Choosing probabilities lower than this will cause the process to die out quickly, while larger probabilities cause the cluster to grow to the size of the entire triangulation, which is also what figure (6) suggests.

8.2 Diameter

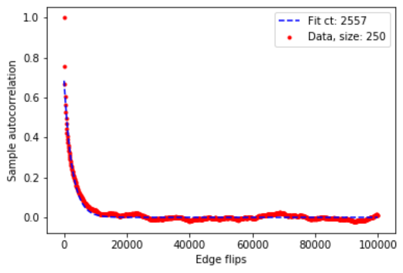
The diameter is the first observable that was studied in this thesis. The sample autocorrelation of the diameter was measured using triangulations of sizes 125, 250, 500, 1000, 2000 and 4000 triangles, in order to determine the scaling of the autocorrelation.
The figures on the left show the edge flip measurements, while the figures on the right show the cluster measurements. All cluster moves were performed with peel probability $p_{peel} = 0.81699$.



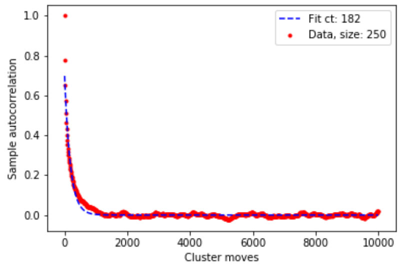
(a) Autocorrelation of edge flips on a triangulation of size 125. Correlation time: 733 flips. Time per flip: $9.0\mu s$



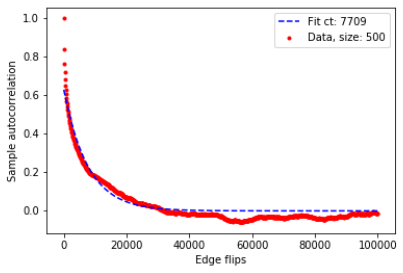
(b) Autocorrelation of cluster moves on a triangulation of size 125. Correlation time: 122 cluster moves. Time per move: $6.8 \cdot 10^2 \mu s$



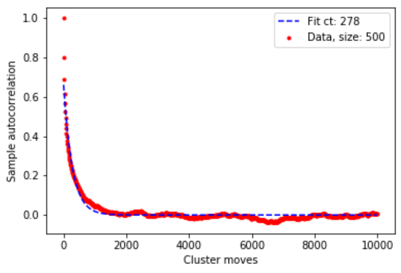
(c) Autocorrelation of edge flips on a triangulation of size 250. Correlation time: 2557 flips. Time per flip: $9.0\mu s$



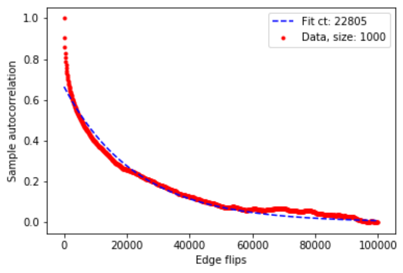
(d) Autocorrelation of cluster moves on a triangulation of size 250. Correlation time: 182 cluster moves. Time per move: $1.4 \cdot 10^3 \mu s$



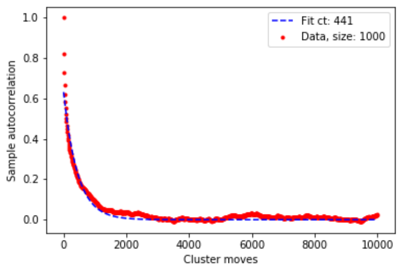
(e) Autocorrelation of edge flips on a triangulation of size 500. Correlation time: 7709 flips. Time per flip: $9.1\mu s$



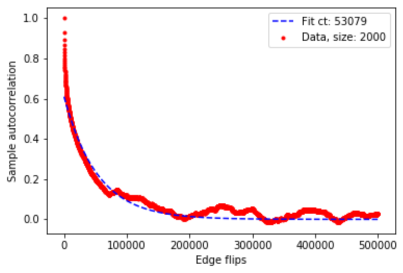
(f) Autocorrelation of cluster moves on a triangulation of size 500. Correlation time: 278 cluster moves. Time per move: $2.9 \cdot 10^3 \mu s$



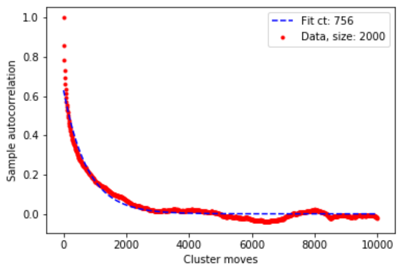
(g) Autocorrelation of edge flips on a triangulation of size 1000. Correlation time: 22805 flips. Time per flip: $9.6\mu s$



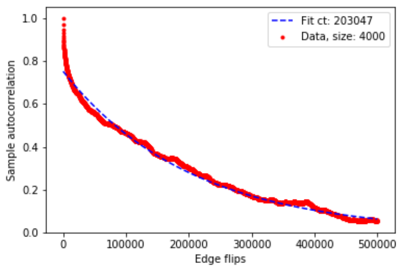
(h) Autocorrelation of cluster moves on a triangulation of size 1000. Correlation time: 441 cluster moves. Time per move: $5.8 \cdot 10^3 \mu s$



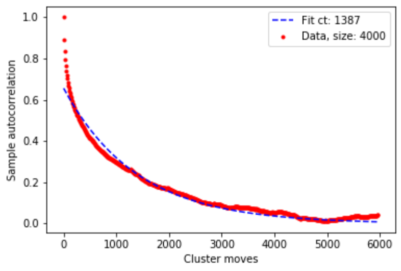
(i) Autocorrelation of edge flips on a triangulation of size 2000. Correlation time: 53079 flips. Time per flip: $11\mu s$



(j) Autocorrelation of cluster moves on a triangulation of size 2000. Correlation time: 756 cluster moves. Time per move: $1.2 \cdot 10^4 \mu s$



(k) Autocorrelation of edge flips on a triangulation of size 4000. Correlation time: 203047 flips. Time per flip: $12\mu s$



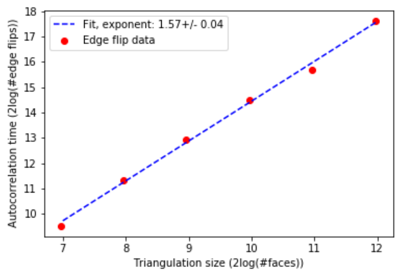
(l) Autocorrelation of cluster moves on a triangulation of size 4000. Correlation time: 1387 cluster moves. Time per move: $2.6 \cdot 10^4 \mu s$

As with all measurements, it is important to get a grasp on the error on the values that were found. In order to approximate these errors, we split the data of each triangulation size up into multiple equally long batches, such that each batch is way larger than the autocorrelation time. We then calculate the autocorrelation time on each batch, which hopefully resembles the original autocorrelation time, and take the standard error of all batches. This results in an approximation on the error. Table (2) shows the correlation times and statistical errors of the diameter measurements, obtained through this method.

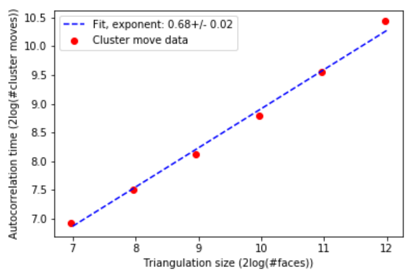
Triangulation size (#faces)	Edge flip autocorrelation time (#flips)	Cluster move autocorrelation time (#cluster moves)
125	733 ± 76	122 ± 10
250	2557 ± 301	182 ± 16
500	7709 ± 924	278 ± 16
1000	22805 ± 4712	441 ± 16
2000	53079 ± 6251	756 ± 22
4000	203047 ± 17882	1387 ± 24

Table 2: Autocorrelation times including an approximation on the error.

In order to see how well the autocorrelation times scale with respect to the size of the triangulation, we plot the values into a log-log plot. Figure (8) shows the results for both the edge-flip move as well as the cluster move.



(a) Scaling of the autocorrelation time using edge flips with respect to triangulation size. All values are taken ^2log. Exponent: 1.57 ± 0.04.



(b) Scaling of the autocorrelation time using cluster moves with respect to triangulation size. All values are taken ^2log. Exponent: 0.68 ± 0.02.

Figure 8: Log-log plots of the diameter observable.

A few things stand out from these measurements. Firstly, the autocorrelation time in the naive edge flips data scales as expected, as measured in [5], with an exponent of 1.57 ± 0.04. Secondly, the result from the cluster algorithm, 0.68 ± 0.02 seems very low, as it is less than half that of the edge flip data. In order to test this value more thoroughly, more and longer measurements would be necessary.

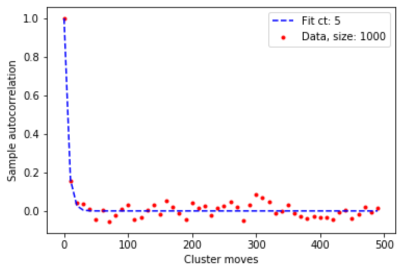
8.3 Laplacian eigenvalue

The second observable, the smallest eigenvalue of the Laplacian matrix of the system, was measured in a similar fashion. The autocorrelation time of this eigenvalue is way lower than that of the diameter, so that we can get proper results using fewer measurements. Figure (9) shows the results of these measurements.

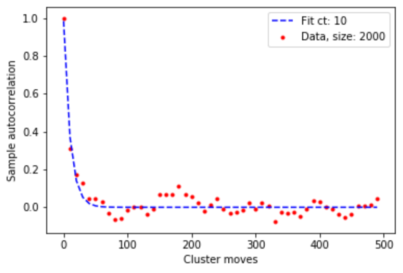
Triangulation size (#faces)	Cluster autocorrelation time (#cluster moves)
1000	5 ± 1 moves
2000	10 ± 1 moves
4000	25 ± 3 moves
8000	118 ± 4 moves
16000	148 ± 17 moves

Table 3: Autocorrelation times including an approximation on the error.

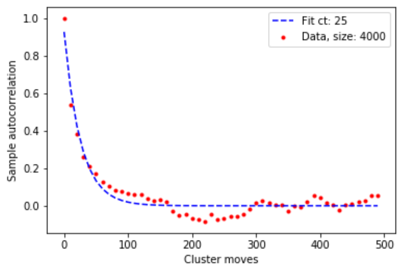
Figure (10) shows the result when fitted using the exponential equation. The resulting value for the exponent is 1.27 ± 0.10. This reveals that the statistical error of the experiment is quite large, as this value is a lot higher than the diameters value. Combining this exponent with the diameter exponent gives a



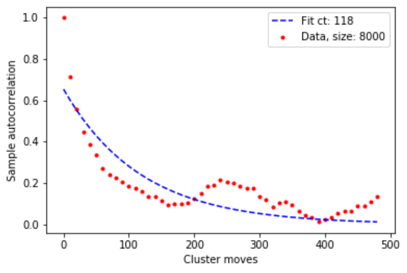
(a) Autocorrelation of cluster moves on a triangulation of size 1000. Correlation time: 5 cluster moves. Time per move: $4.6 \cdot 10^3 \mu s$



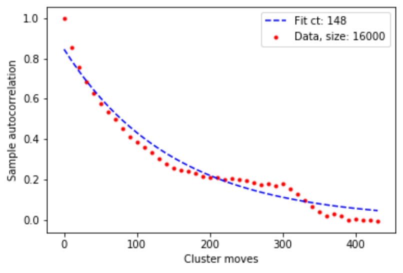
(b) Autocorrelation of cluster moves on a triangulation of size 2000. Correlation time: 10 cluster moves. Time per move: $1.0 \cdot 10^4 \mu s$



(c) Autocorrelation of cluster moves on a triangulation of size 4000. Correlation time: 25 cluster moves. Time per move: $2.4 \cdot 10^4 \mu s$



(d) Autocorrelation of cluster moves on a triangulation of size 8000. Correlation time: 118 cluster moves. Time per move: $5.1 \cdot 10^4 \mu s$



(e) Autocorrelation of cluster moves on a triangulation of size 16000. Correlation time: 148 cluster moves. Time per move: $1.1 \cdot 10^5 \mu s$

Figure 9: Results for the Laplacian eigenvalue observable for different sizes of triangulations.

final exponent with a value of 0.98 ± 0.05, which suggests that the scaling may be linear. However, because of the big difference in the two exponents, this value must be taken with a grain of salt. In order to improve upon this value, a wider variety of observables should be considered.

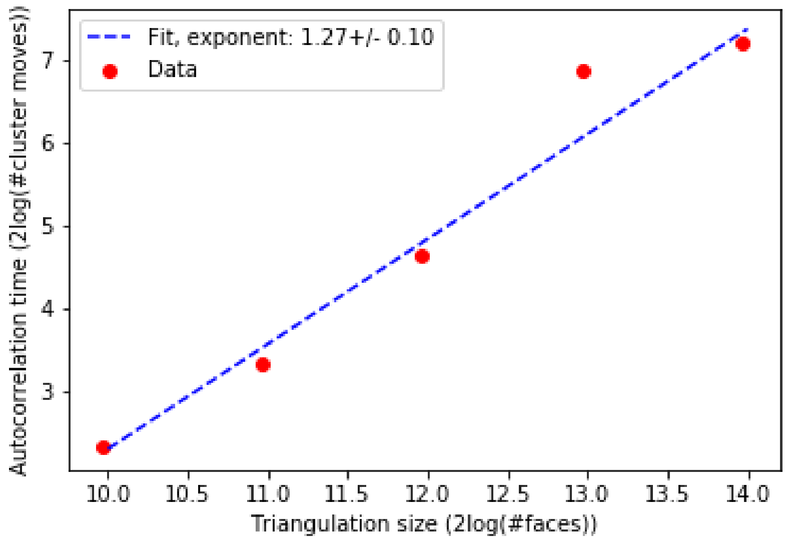
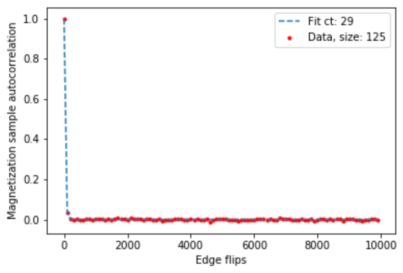


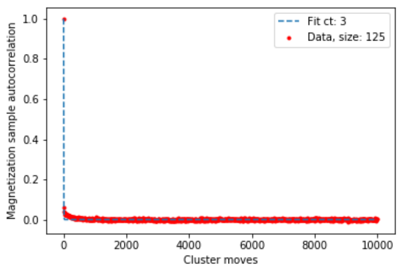
Figure 10: Scaling data for the eigenvalue observable, together with a fit. The fit results in a value of 1.27 ± 0.10 for the critical exponent.

8.4 Magnetisation

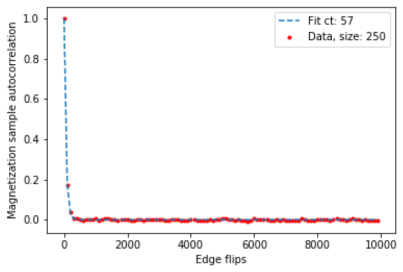
Finally, I present measurements on triangulations with an Ising model coupled to them. In these measurements, I measure both the magnetization as well as the diameter. For the naive measurements, I used both edge flips as well as single spin flips. Figure (11) shows the results for the magnetization measurements, and figure (12) shows the results for the diameter measurements.



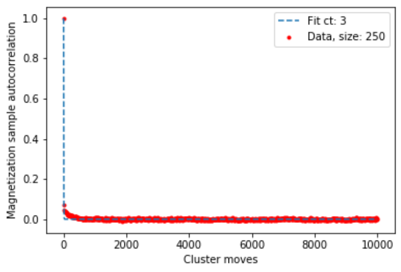
(a) Autocorrelation of edge flips on a triangulation of size 125. Correlation time: 29 flips.



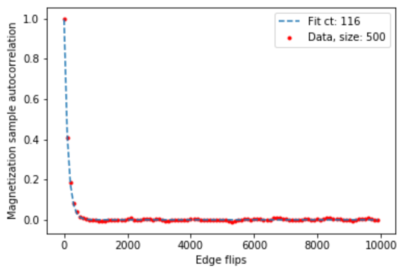
(b) Autocorrelation of cluster moves on a triangulation of size 125. Correlation time: 3 cluster moves.



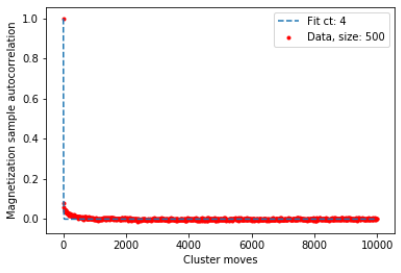
(c) Autocorrelation of edge flips on a triangulation of size 250. Correlation time: 57 flips.



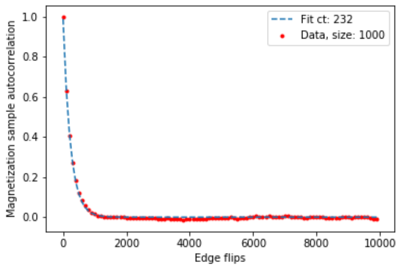
(d) Autocorrelation of cluster moves on a triangulation of size 250. Correlation time: 3 cluster moves.



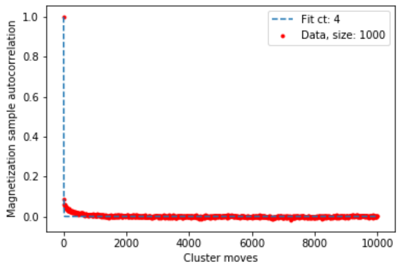
(e) Autocorrelation of edge flips on a triangulation of size 500. Correlation time: 116 flips.



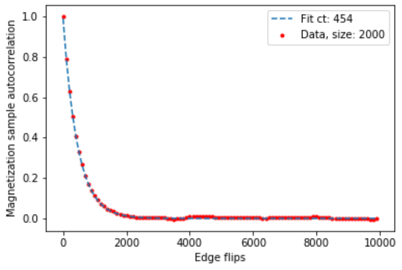
(f) Autocorrelation of cluster moves on a triangulation of size 500. Correlation time: 4 cluster moves.



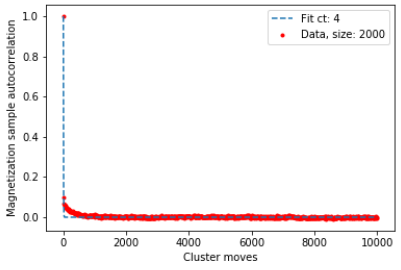
(g) Autocorrelation of edge flips on a triangulation of size 1000. Correlation time: 232 flips.



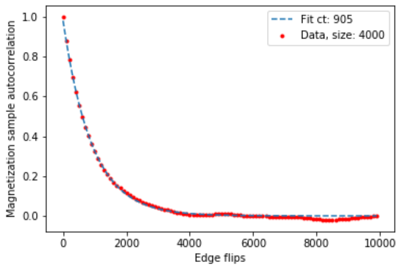
(h) Autocorrelation of cluster moves on a triangulation of size 1000. Correlation time: 4 cluster moves.



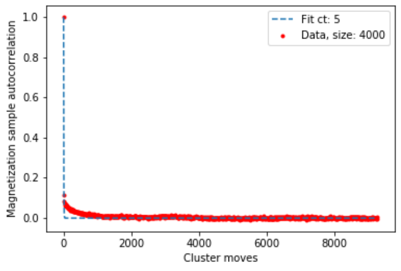
(i) Autocorrelation of edge flips on a triangulation of size 2000. Correlation time: 454 flips.



(j) Autocorrelation of cluster moves on a triangulation of size 2000. Correlation time: 4 cluster moves.

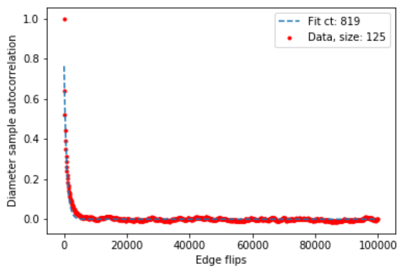


(k) Autocorrelation of edge flips on a triangulation of size 4000. Correlation time: 905 flips.

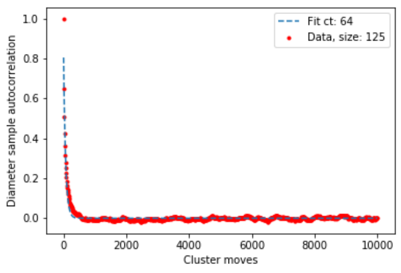


(l) Autocorrelation of cluster moves on a triangulation of size 4000. Correlation time: 5 cluster moves.

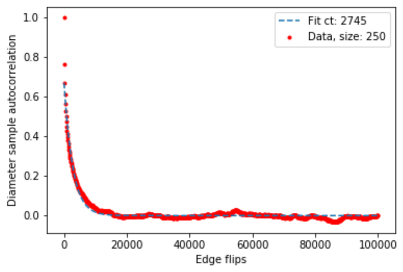
Figure 11: Magnetization measurements on a triangulation.



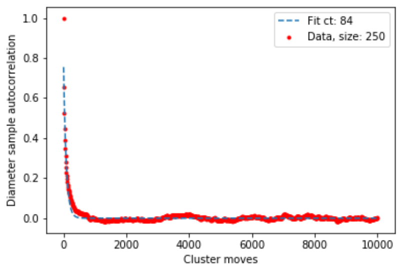
(a) Autocorrelation of edge flips on a triangulation of size 125. Correlation time: 819 flips.



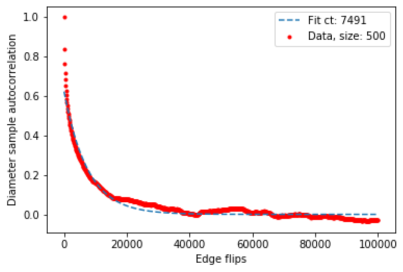
(b) Autocorrelation of cluster moves on a triangulation of size 125. Correlation time: 64 cluster moves.



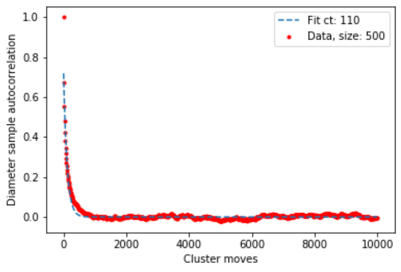
(c) Autocorrelation of edge flips on a triangulation of size 250. Correlation time: 2745 flips.



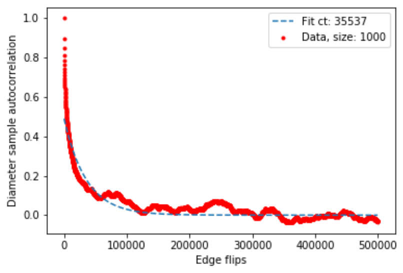
(d) Autocorrelation of cluster moves on a triangulation of size 250. Correlation time: 84 cluster moves.



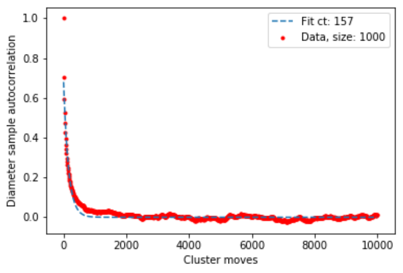
(e) Autocorrelation of edge flips on a triangulation of size 500. Correlation time: 7491 flips.



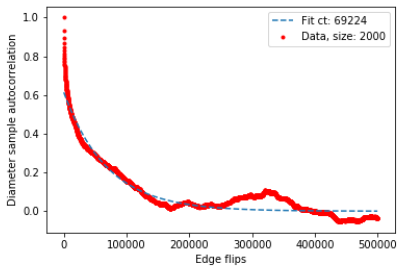
(f) Autocorrelation of cluster moves on a triangulation of size 500. Correlation time: 110 cluster moves.



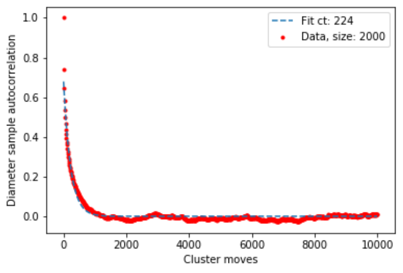
(g) Autocorrelation of edge flips on a triangulation of size 1000. Correlation time: 35537 flips.



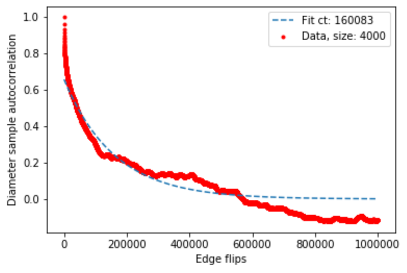
(h) Autocorrelation of cluster moves on a triangulation of size 1000. Correlation time: 157 cluster moves.



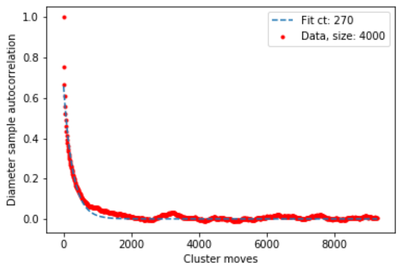
(i) Autocorrelation of edge flips on a triangulation of size 2000. Correlation time: 69224 flips.



(j) Autocorrelation of cluster moves on a triangulation of size 2000. Correlation time: 224 cluster moves.



(k) Autocorrelation of edge flips on a triangulation of size 4000. Correlation time: 160083 flips.



(l) Autocorrelation of cluster moves on a triangulation of size 4000. Correlation time: 270 cluster moves.

Figure 12: Diameter measurements on a triangulation combined with the Ising model.

We analyse the error on the data in the same manner as with the previous measurements, resulting in table (4).

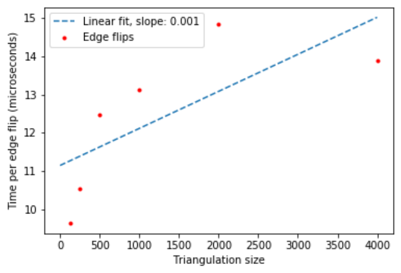
Triangulation size (#faces)	Magnetization autocorrelation time (naive #flips)	Magnetization autocorrelation time (#cluster moves)
125	29 ± 3	3 ± 1
250	57 ± 5	3 ± 1
500	116 ± 10	4 ± 1
1000	232 ± 19	4 ± 1
2000	454 ± 39	4 ± 1
4000	905 ± 81	5 ± 1
Triangulation size (#faces)	Diameter autocorrelation time (naive #flips)	Diameter autocorrelation time (#cluster moves)
125	819 ± 74	64 ± 6
250	2745 ± 446	84 ± 8
500	7491 ± 2166	110 ± 11
1000	35537 ± 4380	157 ± 28
2000	69224 ± 9585	224 ± 26
4000	160083 ± 11618	270 ± 30

Table 4: Results of the measurements on the Ising model coupled to a triangulation, including an error estimate.

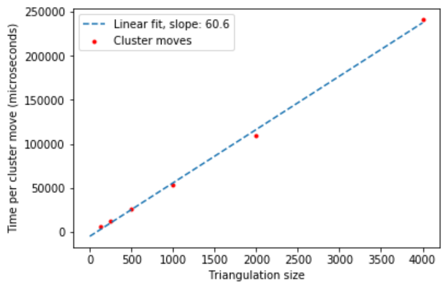
Interestingly, the cluster data does not seem to follow the exponential scaling, whereas the naive data still does. From these few data points, it appears as though the cluster algorithm scales much better than the naive algorithm, but it is too early to draw any conclusions on the exact scaling that it exhibits.

8.5 Wall-clock time

For practical reasons, it is important to measure how long it takes to perform the edge flips and cluster moves. Figure (13) shows the scaling of the wall-clock time as a function of the size of the triangulation. As expected, the edge flips take constant time (we can attribute the slope of the fit to statistical error), and the cluster moves take linear time. Unfortunately, due to the large slope in figure (13b), the cluster algorithm does not outperform the edge flips in many situations.



(a) Scaling of average time per edge flip. A linear fit returns a slope of 0.001.



(b) Scaling of average time per edge flip. A linear fit returns a slope of 60.6.

Figure 13: Average wall-clock times per Markov chain update.

9 Conclusion and outlook

The cluster algorithm appears to scale better than the local edge flip algorithm, when looking at the number of updates necessary to achieve autocorrelation. However, when taking into account the time it takes to perform a move, the edge flip algorithm seems to be the faster algorithm.

The two main obstacles in this research were the statistical errors and the time it takes to perform a cluster move. In the future, the errors can be reduced in several ways. For starters, the number of different sizes considered was quite low in this thesis. The reason for this is that the larger sizes take a long time to complete calculation, due to the linearity of the cluster moves. Considering more sizes results in more data, which should also reduce the error that the least-mean-square fits produce.
Another way to get more accurate results is to consider more observables. However, observables are often difficult to come by, or are computationally expensive.

The other obstacle is the long time it takes to perform a cluster move. This time obviously depends on the machine that the simulation is performed on, but the slope of the scaling may still be brought down. To this end, it may be worthwhile to adapt the algorithm, so that it keeps track of the amount of edges it peels in the building phase. If the number of peels is smaller than some predetermined threshold, it should disregard the cluster. This way, no valuable computing time is wasted on small moves that will not have a big impact anyway. This rejection of the cluster should be done before finding the boundary, so that no $O(N)$ operation is performed when building the cluster (this means that finding the target edge should also be moved to after the building phase).

An interesting follow-up project may be to design another way of generating cycles in the triangulation, and then flipping the cluster. It would be interesting to see if there are more efficient ways of generating these cycles, as opposed to the algorithm studied in this thesis.

Finally, it may be interesting to re-conduct the experiments using different values of the probability p, to see the effect that this has on the autocorrelation time. In this thesis, I made the choice to put p on its critical value, so that the clusters grow linearly with the size of the triangulation. However, the measurements on the Ising model seem to suggest that smaller clusters may lead to smaller statistical errors, and still result in good autocorrelation times. There is also one more situation that I did not try out in this thesis: compare the cluster algorithm on the Ising model to the hybrid algorithm that performs edge flips on the triangulation, but cluster moves on the spins.

Acknowledgements

I would like to thank Timothy Budd for being my supervisor for this bachelor thesis. He found a topic for my thesis that involved both physics as well as computing science, which are two fields in which I have a large interest. It was interesting to work on a project where both of these fields meet.
I would also like to thank Harm Schoorlemmer for wanting to be the second reader.

References

[1] M. Ferrario et al. *Computer Simulations in Condensed Matter Systems: From Materials to Chemical Biology Volume 1. Lect. Notes Phys. 703.* Berlin Heidelberg 2006: Springer. DOI: 10.1007/b11604457.

[2] J. Ambjørn et al. "Nonperturbative quantum gravity". In: *Physics Reports* 519.4-5 (Oct. 2012), pp. 127–210. DOI: 10.1016/j.physrep.2012.03.007. URL: https://doi.org/10.1016%5C%2Fj.physrep.2012.03.007.

[3] Jan Ambjorn. *Elementary Quantum Geometry.* 2022. DOI: 10.48550/ARXIV.2204.00859. URL: https://arxiv.org/abs/2204.00859.

[4] Omer Angel and Nicolas Curien. *Percolations on random maps I: half-plane models.* 2013. DOI: 10.48550/ARXIV.1301.5311. URL: https://arxiv.org/abs/1301.5311.

[5] Anna van Asselt. *Simulating Random Triangulations. Finding a Planck scale in 2d Quantum Gravity Simulations.* 2022.

[6] Dion Bremer. *Source code for the triangulations.* URL: https://gitlab.science.ru.nl/dbremer/dynamical-triangulations/-/tree/main.

[7] T. Budd. *Monte Carlo techniques.* 2022. URL: https://hef.ru.nl/~tbudd/mct/intro.html.

[8] Timothy Budd. *Monte Carlo methods in Dynamical Triangulations.* 2017.

[9] Timothy Budd. *Peeling of random planar maps. Lecture notes for a mini-course given at the Mini-School on Random Maps and the Gaussian Free Field at École normale supérieure de Lyon.* 2017.

[10] Pilu Crescenzi et al. "On computing the diameter of real-world undirected graphs". In: *Theoretical Computer Science* 514 (2013). Graph Algorithms and Applications: in Honor of Professor Giorgio Ausiello, pp. 84–95. ISSN: 0304-3975. DOI: `https://doi.org/10.1016/j.tcs.2012.09.018`. URL: `https://www.sciencedirect.com/science/article/pii/S0304397512008687`.

[11] *Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.* URL: `https://eigen.tuxfamily.org/index.php?title=Main_Page`.

[12] R Loll. "The emergence of spacetime or quantum gravity on your desktop". In: *Classical and Quantum Gravity* 25.11 (May 2008), p. 114006. DOI: `10.1088/0264-9381/25/11/114006`. URL: `https://doi.org/10.1088%5C%2F0264-9381%5C%2F25%5C%2F11%5C%2F114006`.

[13] *Spectra C++ Library For Large Scale Eigenvalue Problems.* URL: `https://spectralib.org/`.