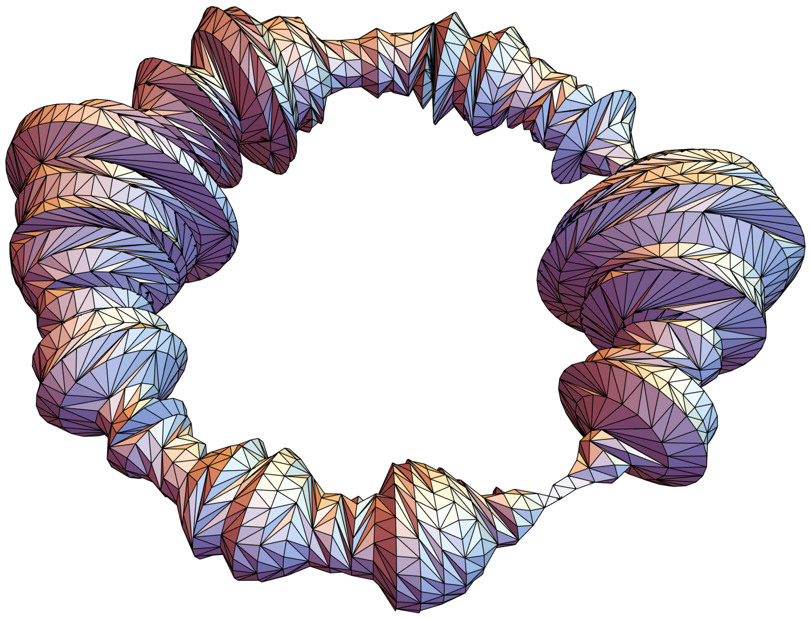
2D Causal Dynamical Triangulations

using Markov-chain Monte Carlo methods

Institute for Mathematics, Astrophysics and Particle Physics, Radboud University

T.B.H. Gerstel J.G.R. van der Duin

January 11, 2022



Abstract

Causal Dynamical Triangulation is a method to regularise the gravitational path integral by building a piece-wise flat spacetime from simplices. The goal of this project is to create an efficient 2D numerical model which can generate these piece-wise flat spacetimes or triangulations, and to use these to measure observables in 2D CDT using Monte Carlo methods to verify the validity of the model by comparison with theory. The simulational model is created in Rust and uses a Markov-chain to generate random triangulations. Data analysis is done in Python. Using this model, we are able to measure the length covariance on a two-dimensional torus for different triangulation sizes of up to 100,000 triangles and demonstrate that the theoretical expectation is within the statistical error our results.

1 Introduction

1.1 Quantum Gravity

A field of theoretical physics which still has many open questions is quantum gravity. One may notice that "quantum gravity" consists of two words: "quantum" and "gravity". The "gravity" aspect is quite well understood. In d dimensions its content can be compactly summarised in the Einstein-Hilbert action

$$S[g_{\mu\nu}] = \frac{1}{16\pi G} \int_{\mathcal{M}} \mathrm{d}^d x \sqrt{-g(x)}(R(x) - 2\Lambda), \tag{1}$$

where the integration is over the entire spacetime manifold $\mathcal{M}$. Here $R(x)$ is the scalar curvature, $g(x)$ is the determinant of the metric tensor $g_{\mu\nu}$, G is Newton's gravitational constant and Λ is the cosmological constant. Classical equations of motion can then be obtained via a variational principle by solving $\delta S = 0$.

A common approach to add the "quantum" aspect to any classical theory is by introducing a path integral. Since in our case the classical action is a functional of the metric $g_{\mu\nu}$, the path integral is given by

$$Z = \int \frac{\mathcal{D}[g_{\mu\nu}]}{\mathrm{Diff}(\mathcal{M})} e^{iS[g_{\mu\nu}]}. \tag{2}$$

But what does this actually mean? How could one explicitly carry out this integration?

1.2 Causal Dynamical Triangulations

We somewhat avoid these questions by adopting a certain regularisation approach, namely Causal Dynamical Triangulations (CDT). Instead of trying to integrate over all continuously curved manifolds, we sum over piecewise flat manifolds that are constructed by gluing a certain class of flat building blocks together. In particular, these building blocks will be d-dimensional regular simplices with a flat Minkowskian interior. These simplices have squared edge lengths fixed at $\pm a^2$ (allowing space- and timelike edges). Here a is the regularisation parameter. The hope is that certain quantities will converge in the limit $a \to 0$. We foliate the manifold into T $(d-1)$-dimensional timeslices, labelled by $t = 0, \dots T-1$. Consecutive timeslices are connected by a single layer of d-dimensional simplices. Different manifolds can be obtained by changing the way simplices are connected. Hence, the functional integration over all metrics turns into a summation over connectivity prescriptions for triangulations. Now the regularised path integral can be written as

$$Z = \sum_{T \in \mathcal{T}} \frac{1}{C(T)} e^{iS[T]}, \tag{3}$$

where the summation is over all triangulations. Here $S[T]$ is the action and $C(T)$ is a symmetry factor for a given triangulation T[1].

Simplifications For simplicity and definiteness, we only consider 2-dimensional spacetimes with periodic boundaries. The resulting topology is that of a 2-torus. With this topology each timeslice is just a ring of spacelike edges. The number of spacelike edges in timeslice t is denoted by $\ell(t)$. With this in mind, we make a few useful observations:

- All triangles have equal volume, so the total volume is proportional to the number of triangles.

[1] Unfortunately the symbol T is the most natural choice for denoting both a particular triangulation, and the number of timeslices contained in such a triangulation. Hopefully it will be clear from context which of the two is used.

- The manifold is piecewise flat, meaning a Wick rotation within each triangle is well-defined.
- In 2 dimensions the integrated scalar curvature is constant and therefore irrelevant for the dynamics.

Using these observations we find that the Euclidean action for a 2-dimensional triangulation T is [2]

$$S[T] = \lambda N(T), \tag{4}$$

where $N(T)$ is the number of triangles, and λ is a dimensionless parameter related to the cosmological constant Λ.

To simplify the symmetry factor $C(T)$ we label all triangles[2]. Each triangulation can be labelled in $N(T)!$ ways, so the Euclidean path integral (or partition sum) becomes

$$Z = \sum_{T_\ell \in \mathcal{T}_\ell} \frac{1}{N(T_\ell)!} e^{-\lambda N(T_\ell)}, \tag{5}$$

where the summation is now over all labelled triangulations. It is possible to split the partition sum into contributions with certain numbers of triangles, as follows

$$Z = \sum_{N=0}^{\infty} \Omega(N) \frac{e^{-\lambda N}}{N!}, \tag{6}$$

where $\Omega(N)$ counts the number of labelled triangulations with N triangles.

1.3 Continuum Limit

Since we are studying a regularised path integral, it is interesting to consider the limit $a \to 0$. In this continuum limit lengths and timespans scale as $L = a\ell$ and $T = at$, respectively[3]. The parameter λ undergoes an additive renormalisation

$$\Lambda = \frac{\lambda - \ln 2}{a^2}. \tag{7}$$

In order to extract information from this continuum limit, it would be useful to have an expression for the continuum propagator. In this context, the propagator is proportional to the probability that a spacelike slice of length L_1 evolves into a slice of length L_2 over some time interval of size T. Fortunately, a closed form for this propagator has been found, and it is given by the beautiful expression [1]

$$G_\Lambda(L_1, L_2, T) = \frac{e^{-\left[\coth\sqrt{\Lambda}T\right]\sqrt{\Lambda}(L_1+L_2)}}{\sinh\sqrt{\Lambda}T} \frac{\sqrt{\Lambda L_1 L_2}}{L_2} I_1\left(\frac{2\sqrt{\Lambda L_1 L_2}}{\sinh\sqrt{\Lambda}T}\right), \tag{8}$$

where $I_1(x)$ is a modified Bessel function of the first kind.

To be able to extract information from this propagator, we consider the limit $\sqrt{\Lambda}T \gg 1$. In this regime, we can approximate the propagator as

$$G_\Lambda(L_1, L_2, T) \approx 4\Lambda L_1 e^{-\sqrt{\Lambda}(L_1+L_2+T)}. \tag{9}$$

In the limits $\sqrt{\Lambda}T_1, \sqrt{\Lambda}T_2 \gg 1$, the probability distribution[4] for the length L is given by

$$p(L) = \frac{1}{Z} G_\Lambda(L_1, L, T_1) G_\Lambda(L, L_2, T_2) = 4\Lambda L e^{-2\sqrt{\Lambda}L}, \tag{10}$$

[2]This is of course also convenient when representing the triangulation in a computer.

[3]Indeed, unfortunately there is yet another quantity for which T is the most natural (and conventional) choice.

[4]Note that due to the large time regime we restrict ourselves to, we expect this distribution to be independent of the time at which L is considered.

where Z is included to ensure proper normalisation. The mean of this distribution is

$$\langle L \rangle = \frac{1}{\sqrt{\Lambda}}. \tag{11}$$

And its variance is given by

$$\sigma_L^2 = \langle L^2 \rangle - \langle L \rangle^2 = \frac{1}{2\Lambda}. \tag{12}$$

The quantities we have discussed so far only describe the global behaviour of L. While these provide useful checks on our model, we would also like to consider quantities that show how different lengths are related in time. In particular, we would like to compute the autocovariance of L as a function of T. To this end we compute (in the limits $\sqrt{\Lambda}T_1, \sqrt{\Lambda}T_2 \gg 1$ but with $\sqrt{\Lambda}T = \mathcal{O}(1)$) the joint probability distribution

$$p(L(0), L(T); T) = \frac{1}{Z} G_\Lambda(L_1, L(0), T_1) G_\Lambda(L(0), L(T), T) G_\Lambda(L(T), L_2, T_2), \tag{13}$$

where $T > 0$ and again Z ensures proper normalisation. From this probability distribution we can compute the expectation value

$$\langle L(0)L(T) \rangle (T) = \frac{e^{-2\sqrt{\Lambda}|T|}}{2\Lambda} + \frac{1}{\Lambda}, \tag{14}$$

where the absolute value comes from the fact that by symmetry T and $-T$ should give the same result. Then we find that the autocovariance is

$$\gamma_L(T) = \langle L(0)L(T) \rangle - \langle L(0) \rangle \langle L(T) \rangle = \frac{e^{-2\sqrt{\Lambda}|T|}}{2\Lambda}. \tag{15}$$

The main goal of this project is to measure these quantities on the discrete model. In order to compare discrete and continuum quantities, one needs to make the substitutions $L = a\ell$ and $T = at$. The observables are rescaled in the obvious way as $\sigma_L = a\sigma_\ell$ and $\gamma_L = a^2\gamma_\ell$, such that all factors of a match. The cosmological constant Λ can be rescaled using Eq. (11), resulting in

$$\Lambda = \frac{1}{a^2 \langle \ell \rangle^2}. \tag{16}$$

However, in our implementation of the discrete model $\langle \ell \rangle$ will be fixed, and its value must be provided as an input parameter[5] L (See section 2.3).

2 Methods

2.1 Markov Chain Monte Carlo

The goal is now to extract information from the distribution defined by (6). Expectation values of observables can be sampled using a Monte Carlo approach. Suppose we sample n triangulations $T^{(i)}$ from (6). The expectation value of an observable $f(T)$ can then in theory be obtained from

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(T^{(i)}) \overset{d}{=} \langle f(T) \rangle, \tag{17}$$

[5]Yet another unfortunate naming convention: in this context L is a discrete parameter provided as an input for the model, whereas before it denoted the continuous length of some spacelike slice.

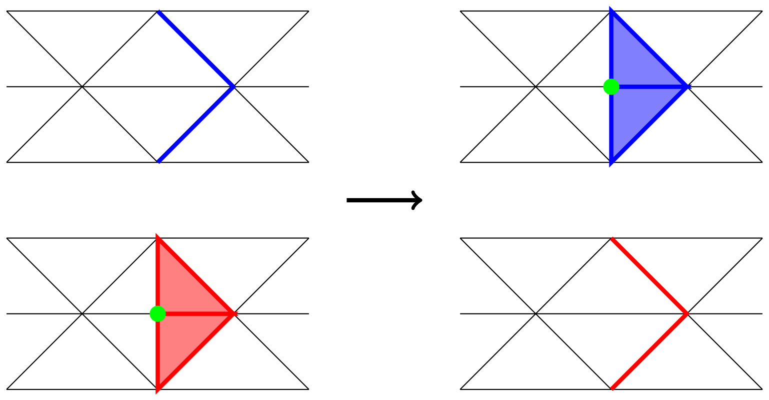


Figure 1: Shard move update rule. Shard source in red (bottom), destination in blue (top). Initial (left) and final (right) states. Order 4 vertices affected by this move are marked with a green dot.

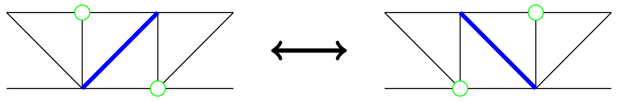


Figure 2: Flip move update rule. The flipped edge is shown in blue. Order 4 vertices (if they exist) are marked with a green circle.

where the d above the equality denotes that this is a convergence in distribution. Directly sampling triangulations is not viable. Instead, we use a Markov Chain to explore the space of all possible triangulations. Hence, the label Markov Chain Monte Carlo (MCMC) for our approach.

Sampling from (6) is not practical. It turns out that $\Omega(N) \sim N!2^N$. This means that for $\lambda > \ln 2$ large volumes are suppressed so that a typical triangulation will contain only a handful of triangles. However, when $\lambda < \ln 2$ the partition sum (6) diverges and the problem is ill-defined. A solution is to consider only triangulations of a certain fixed volume at a time. An advantage of this approach is that the desired distribution becomes uniform, since the weight of each triangulation depends only on its volume. In practice, this can be achieved by using Markov Chain updates that keep the number of triangles fixed.

2.2 Update rules

Ergodicity An important property of a triangulation is how its volume is distributed over different timeslices. It therefore makes sense to choose an update rule that can change the length $\ell(t)$ at some time t. However, because we wish to fix the volume this change in length should be compensated elsewhere. Thus, we choose a move that takes an edge and accompanying triangles (we refer to such objects as "shards") from some source location, and moves it to some destination. This "shard move" is shown schematically in Fig. 1. However, this move alone is not enough to be able to reach all possible triangulations. Thus, we introduce another move that can flip the diagonal between two triangles. This "flip move" is shown schematically in Fig. 2. According to [2], this combination of update rules is ergodic.

Detailed Balance But this is not the whole story. Using these update rules we can now sample all triangulations of a given volume, but can we do so *uniformly*, as required by (6)? A common way to ensure this is to enforce detailed balance. Since in this case the desired distribution is uniform, this amounts to enforcing

$$p(T \to T') = p(T' \to T), \tag{18}$$

for any two triangulations T and T'. Here $p(T \to T')$ is the probability to go from T to T' in a single Markov Chain step.

First consider the shard move. In order to be able to remove a shard, we need a vertex of order 4 (i.e.: a vertex with 4 edges)[6]. Start by uniformly choosing such a vertex. To be consistent, we always remove the shard to the right of the selected vertex. Then we uniformly choose a spacelike edge (which is not the edge in the shard) and insert the shard to the left of it. This is done in such a way that an order 4 vertex is created between the selected and inserted edges (see Fig. 1). The probability of performing a particular instance of this move is then[7]

$$p_{\text{shard}}(T \to T') = \frac{1}{V_4(N/2-1)}, \tag{19}$$

with V_4 the number of order 4 vertices, and N the number of triangles. Note that this move preserves V_4: One order 4 vertex is always destroyed where the shard is taken, and one is always created where it is inserted. Any surrounding vertices are not affected. Since N is also constant, the inverse move will have identical probability and detailed balance is satisfied.

Now consider the flip move. This move is possible where neighbouring triangles have opposite orientation. First uniformly choose a triangle. To be definite, consider its neighbour to the right. If these two triangles have opposite orientation, perform the flip. If this is not the case, reject the move but still count it as a step in the Markov Chain. The probability of performing a particular instance of this move

$$p_{\text{flip}}(T \to T') = \frac{1}{N}. \tag{20}$$

Because N is fixed it is clear that this move obeys detailed balance. Note that since both these moves independently satisfy detailed balance, they can be performed in any sequence. See section 2.3 for more details on how these two moves are combined.

2.3 Implementation

Data Structures How can one actually represent a triangulation inside a computer? There are of course multiple ways to approach this. By far the most frequent operation in the model is performing Markov Chain steps. It therefore makes sense to choose a data structure in which it is easy (i.e. fast) to execute these steps. To facilitate this, we choose to store only adjacency information. Because of this, the computation time of executing a Markov Chain step is constant, and does not scale with the system size. However, the computation time of doing measurements does scale linearly with N. But this is not so bad, since measurements are only performed in a fraction of all Markov Chain steps.

In particular, we keep a list of all `triangles`. Then for each `Triangle` we store three labels corresponding to each of their neighbours (neighbours to the `left`, `right` and in the `time`like directions), as well as its `Orientation` (whether it points `Up` or `Down`). Finally, for quick access a list of `order_four` vertices is stored (labelled by the `Triangle` northwest of the vertex). All of this information is stored in the `Universe` structure. In pseudocode the `Universe` is defined as:

```
Universe = {
    triangles: [Triangle],
    order_four: [int],
}
```

[6] If you are worried that such a vertex might not always exist, you are correct. However, if this is the case we can simply perform the flip move instead.

[7] Note that here T and T' must be two triangulations that are related by a single application of a shard move. If they are not, the probabilities vanish in both directions and detailed balance is satisfied automatically.

And the `Triangle` structure is defined as (in pseudocode):

```
Triangle = {
    orientation: Orientation,
    time: int,
    left: int,
    right: int,
}
```

Note that we use `int`egers to label `Triangle`s by index in the list of `triangles` stored in `Universe`.

With these data structures the implementation of the moves is relatively straightforward. Uniformly sampling a `triangle` or `order_four` vertex can be done by simply generating a random `int`eger between 1 and the length of `triangles` or `order_four`, respectively. Sampling a spacelike edge is achieved by sampling a `triangle` and selecting its spacelike edge.

In order to actually execute the shard move, six pairs of neighbours need to be reassigned, a single `order_four` vertex is moved, and surrounding vertices need to be checked on whether they are (still) `order_four`[8]. For the flip move, two pairs of neighbours need to be updated. There are two vertices that might become `order_four`. Additionally, there are two that, if they were `order_four` before the move, they certainly no longer are afterwards. These four vertices are marked with green circles in Fig. 2, and they should be checked explicitly.

Length Measurements So far we have focussed on how the update rules are implemented. However, another very important aspect of the simulation is doing measurements. Perhaps the most obvious thing one can measure on a triangulation is the volume of each timeslice. In our case, each timeslice is a one-dimensional ring, so its volume (or length) can be described by the number of edges (or equivalently the number of vertices) it consists of. Doing this for every timeslice t results in a *length profile* $\ell(t)$. All other observables considered in this work are derived from $\ell(t)$, see section 2.4 for more details.

But how can this measurement be performed on our `Universe`? First we choose (it does not really matter how) a `Triangle` to use as reference. We keep walking to the `right` until we reach the original `Triangle` again. The amount of `Up`ward pointing `Triangle`s we passed is then the length $\ell(0)$ of the first timeslice, specifically of the lower boundary timeslice as we count length by the amount of spacelike edges. We can then move to the next timeslice by going to the `time`like neighbour of a `Down`ward pointing `Triangle`. This is repeated until we encounter the first `Triangle` again.

Visualisation Besides being able to inspect the length profile, it is also interesting to be able to visually inspect the triangulation directly. To be able to do this we recreate the triangles from vertices with coordinates to create a mesh which can then be encoded in a standard computer graphics mesh format or directly rendered by Wolfram Mathematica, as we did.

To do this, a list of vertices needs to be created which are linked to the triangles they form the corners of. This is done similarly to the length profile measurements. We start at a `Triangle`, label all three vertices and store the triplet of the vertices making up the triangles in clockwise fashion. Using `right` we go to the next `Triangle`, label the new vertex and store the new triplet of vertices. This is repeated until we walk back to the original triangle of this layer. Then we move to the next layer by using a `time`like neighbour of a `Down`ward pointing triangle, and repeat the process. Note that for the layers after the first the bottom vertices have already been labelled. This is repeated

[8]If the order 4 vertices were labelled by the triangles northeast of them, this would not be necessary. In that case the order 4 vertex would automatically move with the shard to its new location. The surrounding vertices also automatically remain correctly in or out of the order four list. Unfortunately we only realised this near the very end of the project.

until the last layer before one is back in the original layer. In performing this labelling, one needs to be careful that the same vertex never gets relabelled such that all `Triangle`s connected to the vertex are actually associated to the same label. Then we chose to use a cylindrical visualisation, which makes a cut in the real toroidal topology of the triangulation. So instead of associating the first and last timeslice with each other, we simply give the vertices in the first timeslice (the lower boundary of the first layer) and the vertices last timeslice (the upper boundary of the last layer) different labels. One can visualise the full triangulation of the toroidal topology without creating a cut, but then one needs to be really careful to make sure the labels of the vertices of the last timeslice correspond to the correct vertices.

After the list of vertex labels and the triplets of these labels have been created, the coordinates of the vertices need to be determined, also called choosing the *embedding*. We choose a timeslice preserving cylindrical embedding, where the vertices of a timeslice are equally spread around a circle with a circumference reflecting the length of that timeslice and subsequent timeslices are placed parallel to one other a corresponding distance away. This has the result that all spacelike edges are roughly the same length as one other and as the distance between timeslices. The relative rotation of the timeslices is chosen such that one of the timelike edges has the shortest distance possible between the timeslices; an alternative choice of rotation is to choose it such that the average length of the timelike edges is minimal. Using this method we can produce visualisations of the triangulations like the one displayed in Fig. 3.

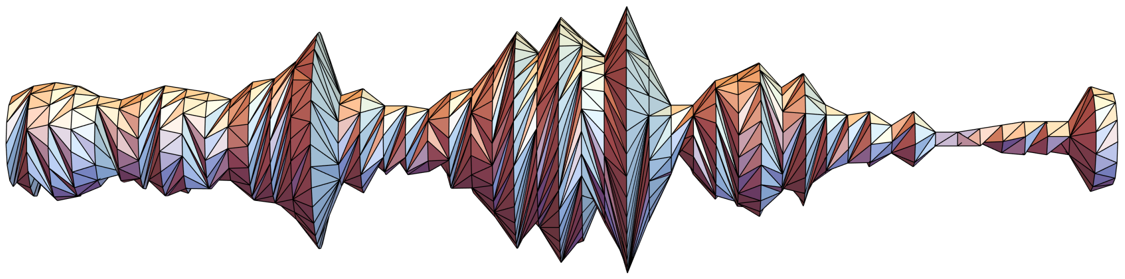


Figure 3: A visualisation of a small triangulation using a timeslice preserving cylindrical embedding

Command-line Interface In order to actually run a simulation, we of course need to provide some concrete parameter values. The most obvious ones are related to the system size. The number of triangles is fixed, so that would seem like a logical input parameter. However, instead we choose to provide the number of timeslices T and the initial length per timeslice L as input. In the initial state, there are T timeslices which each have length L. Furthermore, triangles alternate between upward and downward pointing triangles, resulting in a flat spacetime. An example of this initial state is shown in Fig. 4. The total number of triangles is then $N = 2TL$. This makes it easier to change T and L independently of each other.

Another input parameter has to do with the fact that we are using two different types of Markov Chain moves. We need some way of choosing between these two moves. Therefore, we introduce

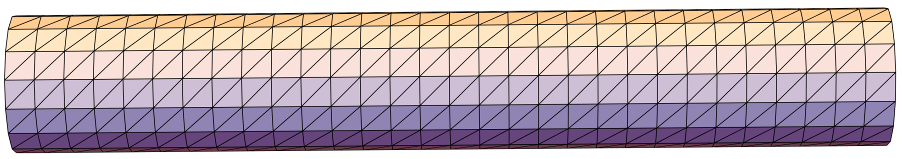


Figure 4: Flat initial state of the triangulation.

a move rate r. In particular r is the probability that any given move will be a shard move. Note that introducing this rate r does not affect detailed balance: we simply multiply both sides of the detailed balance equation by r (for the shard move) or $1 - r$ (for the flip move). This move ratio can be set separately for the equilibration phase and the measurement phase.

Finally, there are some parameters that are related to the Monte Carlo measurements. The obvious one is of course which observable is measured: the whole length profile $\ell(t)$ or only its standard deviation σ_ℓ. Other parameters include the number of measurements to take, the number of sweeps[9] in-between measurements, the length of the equilibration phase and a path to which the data is written.

2.4 Observables

Finding good observables to measure is quite challenging in (C)DT. This may partly be due to a lack of experimental data to compare simulations to. But also, where many Monte Carlo simulations in physics are concerned with describing the behaviour of fields on a lattice, naturally giving some combination of the field values as observables, there is no such field in CDT, but the lattice itself should provide the information of interest.

The chosen observables should naturally not depend on any details of the implementation, like the use of labelling in our implementation. And for an observable to be of physical interest it should also have a well-defined limit for the amount of triangles $N \to \infty$. Observables that are often looked at are different notions of dimension, notably the *spectral dimension* [2] and *Hausdorff dimension* [1, 2], the latter of which has also been discussed in the case of Dynamical Triangulations in the course Monte Carlo Techniques. And a newer observable of interest is a certain notion of *Quantum Curvature* – a sort of extended notion of Ricci curvature which can be applied to the used triangulations and obeys a proper limit [3].

However since the focus of this project was more the implementation of a Markov Chain Monte Carlo simulation, and the measurements and analysis of these observables is somewhat involved, we only consider observables that are derived from the volume of the space-like slices. This means that in the case of 2D CDT we look at the *length profile* $\ell(t)$ as explained earlier.

Length profile The length profile itself still depends on a certain choice of origin in t, while we consider a toroidal topology, so there is no preferred origin. Thus, any observable should have some sort of averaging over t.

The simplest observable to consider is the average of $\ell(t)$, but this is trivially L since the total amount of timeslices and triangles and thus vertices is kept fixed. So the simplest non-trivial observable we can think of is the standard deviation σ_ℓ of the length profile $\ell(t)$:

$$\sigma_\ell^2 = \frac{1}{T} \sum_{t=1}^{T} \Big(\ell(t) - L\Big)^2, \tag{21}$$

using the usual definition of the population variance.

But σ_ℓ does not contain any information about the relation of lengths between different times t. So an arguably more interesting observable is what we call the *length covariance*:

$$\rho_\ell(t) = \frac{1}{T} \sum_{t_0=1}^{T} (\ell(t_0) - L)(\ell(t_0 + t) - L), \tag{22}$$

note that $\rho_\ell(0) = \sigma_\ell^2$. Important to note is that the length correlation is still a function of t but this t is the time difference between two time-slices, so there is no origin dependence.

[9]A sweep is defined as $2TL$ Markov Chain steps.

3 Results and Discussion

The observables we measured in the $1+1$D CDT model are the *standard deviation* of the length profile σ_ℓ and the *length covariance* $\rho_\ell(t)$ as introduced in section 2.4. In this section we will present and discuss the results found for these observables.

3.1 Pre-analysis

Before the actual measurements can be performed some data analysis need to be done.

Equilibration To be able to take measurements of the wanted observables in the Markov-chain Monte Carlo simulation the system needs to be thermalised. Which is to say that the system should be in a "typical" state, such that the expectation value of an observable at any timestep in the simulation is the same as that of any other. We start the system in a non-typical, flat spacetime, so it takes some Markov-chain steps before the system is in equilibrium. We want to only start measurements after this *equilibration time*, so we need to estimate it to know when to start measuring.

Preferably, one uses the observable of interest to determine the equilibration time, however it is very difficult to quantify when a function like $\rho_\ell(t)$ has thermalised. So for both observables we determine the equilibration time using σ_ℓ. To determine when the system has equilibrated in terms of σ_ℓ we fit a function which converges exponentially towards the average value: $\sigma(t) = \hat{\sigma}\left(1 - e^{t/t_{\text{eq}}}\right)$, where t here is the Monte Carlo simulation time, and t_{eq} then defined equilibration time. This works well because initially $\sigma_\ell = 0$ as the starting triangulation is flat, which over time changed to fluctuating around some mean value. This fitting procedure is visualised in Fig. 5. So using this

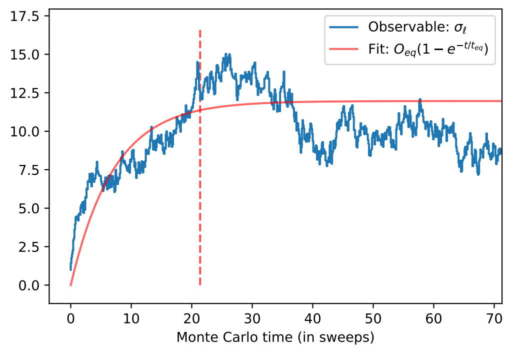


Figure 5: Visualisation of determination of thermalisation by fitting an exponential convergence. *Marking at* $3t_{eq}$

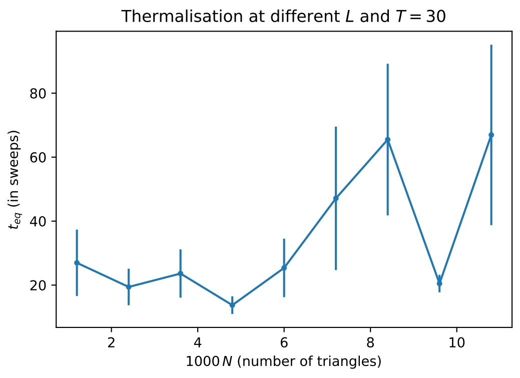


Figure 6: Equilibration time for different sizes of L, based on 10 samples for each system size. Determined at move rate $r = 0.4$.

method we can estimate the equilibration time given a trace of σ_ℓ.

Then instead of determining the equilibration time for every system we wish to measure, we attempted to determine the dependence of the equilibration time on different system sizes. To do this we measured t_{eq} for different values of L and repeated each measurement 10 times to obtain more accurate results and get an estimate of the error in the results. This yields the results in terms of sweeps presented in Fig. 6 (note that $N = 2LT$).

The found results show no clear dependence of t_{eq} on the system size, and since we only wish to obtain an order of magnitude estimate, it seems reasonable to assume that *in terms of sweeps* the equilibration time is independent of the system size. We then take the equilibration time to be $t_{\text{eq}} = 200\,\text{sweeps}$ for all simulations to be safe. This is a very crude approximation of the equilibration time, but since 200 sweeps is relatively small it seems unnecessary to put more work in a better estimate of the equilibration time.

Autocorrelation Once the system has thermalised, we can start to take measurements of the observables of interest. However, after a single simulation timestep the newly obtained state is still very similar to the previous state, thus observables measured in these different states are by no means independent; they are in fact highly correlated. Having many correlated measurements is not useful as they do not make the final estimate better, and take up a lot of unnecessary space and computation time. Moreover, to be able to make and estimate of the error in the final results, it is crucial to know the correlation between measurements.

To estimate the correlation time we will again use σ_ℓ as observable, as this is much easier than using the function $\rho_\ell(t)$. The correlation time is then estimated as usual by determining the *autocorrelation* of the standard deviation observable over time:

$$\rho(t) = \frac{1}{Z} \sum_{i=1}^{M-t} (\sigma_{\ell,i} - \bar{\sigma}_\ell)(\sigma_{\ell,i+t} - \bar{\sigma}_\ell),$$

where $\sigma_{\ell,i}$ is the ith measurement in a set of M measurements of σ_ℓ, $\bar{\sigma}_\ell$ is the sample average, and Z is a normalisation factor such that $\rho(0) = 1$.

Now for long enough measurements we expect that the autocorrelation follows exponential decay with which we can define the correlation time t_{cor}, such that $\rho(t) \approx \exp(-t/t_{\text{cor}})$. So to estimate t_{cor} we simulate a long trace of σ_ℓ and fit an exponential decay to the autocorrelation of that trace. To estimate the error on the estimate of the correlation time we can repeat the measurements several times; or we can simulate a very long trace and divide the trace up in batches which we treat as separate measurements, which simply saves on thermalisation.

We would like the autocorrelation to be as small as possible. So we can tune the introduced move rate r such that the autocorrelation is minimized. To achieve this we estimate the autocorrelation for different move rates, the results of which are displayed in Fig. 7. From this figure one can clearly

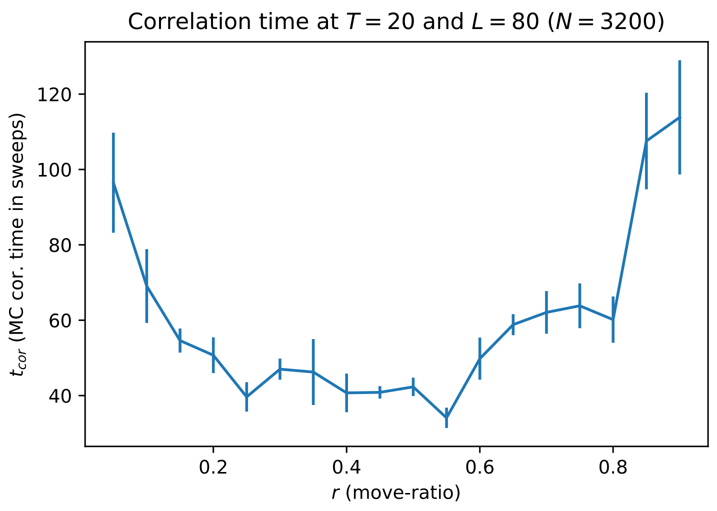


Figure 7: Correlation time as function of the move rate.

see that move rates close to 0.0 or 1.0 have a very long correlation time. This makes sense as only

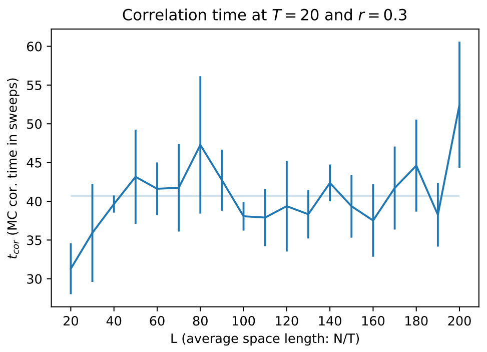


Figure 8: Correlation time at $T = 20$ for different L, with corresponding fit.

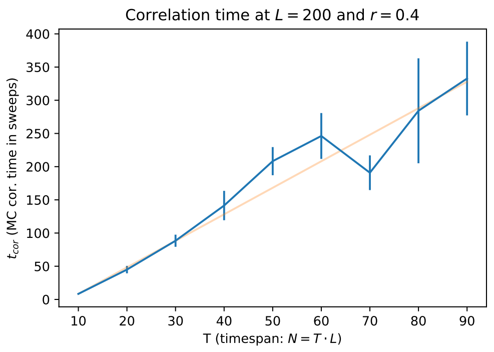


Figure 9: Correlation time at $L = 200$ for different T, with corresponding fit.

flip moves cannot change the length profile at all and without flip moves there are no new order 4 vertices created for shard moves to occur, so the shards move around between the same places. It also appears that the correlation time is not very sensitive to the move rate r, as the correlation time estimate remains roughly constant for r between 0.3 and 0.5. So it seems safe to simply pick $r = 0.4$ as there is no need to tune the rate to a very specific value.

Then finally we wish to estimate the dependence of the correlation time on the system size. To do this we estimate the correlation time for multiple system sizes like was done for the equilibration time. We measured the correlation for different values of T and L keeping the other constant. The results of these measurements are presented in Fig. 8 and 9. From Fig. 8 it seems that *in terms of sweeps* the correlation time is roughly constant under L and can at $T = 20$ taken to be $t_{\text{cor}} = 41$ sweeps. However, under changing T the t_{cor} seems to be growing roughly linearly, giving the crude approximation $t_{\text{cor}} \approx 4\,(T - 10)$ that can be use as an estimate for the correlation time for a wanted system size. This estimate is very crude, but keep in mind that the final results do not actually depend on this measurement of the correlation time. It helps to know the correlation time to be able to only save relevant data, but it will not affect the final results. So this approximation is sufficient for the purposes of this simulation.

Note that this linear scaling in T of the correlation time (in terms of sweeps), means that the amount of Markov-chain Monte Carlo steps grows like LT^2, or alternatively like $N^{3/2}$ for a constant ratio T/L. This scaling means that our computation time will increase like $N^{3/2}$ the system size is increased, and puts a limit on the system sizes that can be measured.

3.2 Measurements

With the system in equilibrium and an estimate for the correlation time, we can start to measure observables. We wish to compare the measured standard deviation and covariance to the theoretical expectation of 2D CDT, as determined in the $T \to \infty$ limit. To simulate measurements in this limit we wish to choose T such that $L \ll T$; in this case we use $T = 20\,L$ as larger values of T have a large impact on computation time. We can then compute the *length profile* $\ell(t)$ over a range of L values. Within time constraints, we were able to measure a range of $L = 15, 20, \ldots, 50$; we performed 100 measurements for each L at $4L = \sqrt{0.4\,N}$ sweep intervals, which is chosen as the correlation time appears to be roughly $t_{\text{cor}} \approx 3\sqrt{N}$ sweeps[10] at this T/L ratio.

[10]This is different from the estimate of t_{cor} determined before, this discrepancy is further discussed in 3.3

Standard deviation The data from the simulation can be analysed to obtain estimates of σ_ℓ. In principle, it would be possible to estimate Λ based on either the mean length (using Eq. (11)) or the variance σ_ℓ (using Eq. (12)). In the end, we choose to use the latter, as will be explained in the text later on. From Eq. (11) we expect the cosmological constant and equivalently the standard deviation to depend on the system size like[11]:

$$\Lambda = \frac{1}{L^2}, \qquad \langle \sigma_\ell \rangle = \frac{L}{\sqrt{2}} \approx 0.71\, L. \tag{23}$$

To estimate σ_ℓ and the error in the estimation the standard deviation is computed from the length profile, according to Eq. (21), for every measurement after which *batching*[12] is used. For these measurements 10 batches were used, and the batches are checked to have no significant auto-correlation. The results of this analysis for the different values of L is presented in Fig. 10 together with a power-law fit ($\sigma_\ell = a\, L^\nu$).

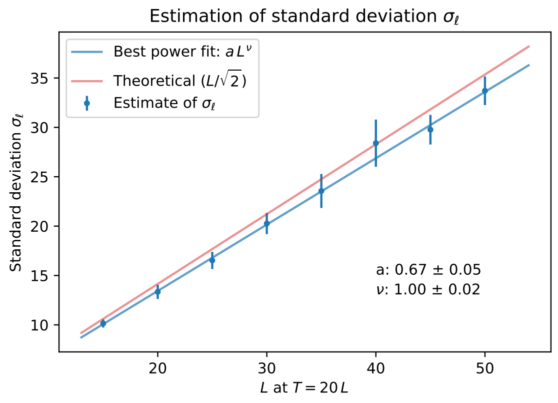


Figure 10: Measurement results of σ_ℓ for different system sizes L, with power-law fit and theoretical expectation.

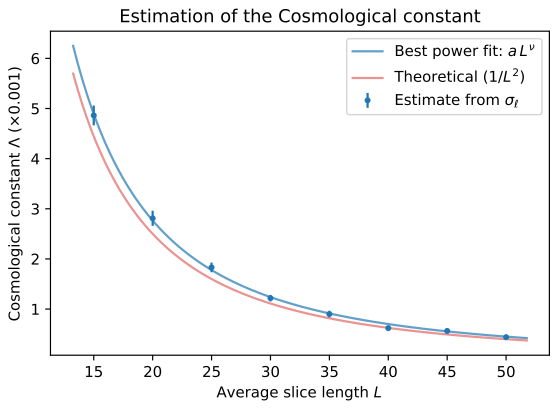


Figure 11: Estimates for Λ based on the σ_ℓ measurements, with power-law fit and theoretical expectation.

For the fit is found that $a = 0.67 \pm 0.05$ and $\nu = 1.00 \pm 0.02$, thus we see that as expected by Eq. (23) σ_ℓ indeed grows linearly in L with good accuracy and 0.71 falls within the 68% confidence interval of a. However, from the plot in Fig. 10 it seems that the measurements systematically underestimate the standard deviation, as is discussed in section 3.3.

From the estimates of σ_ℓ we can also estimate the cosmological constant Λ of the simulated systems, which is used later. This is done simply by using the estimator $\Lambda = 1/2\sigma_\ell^2$, and determining the mean value and error accordingly. The resulting Λ estimates for different L are displayed in Fig. 11. From this no real additional information can be extracted as it is simply a transformation, but we can confirm that the cosmological constant closely relates to L with the expected relation (23).

Length covariance Next we can also analyse the *length covariance* $\rho_\ell(t)$ from the measured length profiles, and compare them to the theoretical covariance (15), now using the discrete t and Λ instead. We compute the covariance for every measurement according to Eq. (22) and use the

[11]In this context Λ refers to what the cosmological constant would be for $a = 1$. A more correct but clumsy notation would be $a^2\Lambda$.

[12]In *batching* correlated measurements are divided in M batches larger than t_{cor} and each batch is averaged to obtain M uncorrelated measurements that are considered *independent identically distributed*. The observable is then estimated by taking the mean of the obtained measurements, and the error is estimated by the standard deviation of the obtained measurements divided by $\sqrt{M-1}$.

same *batching* method to estimate the covariance and error at every t. The resulting estimates are displayed in Fig. 12 with 68% confidence intervals.

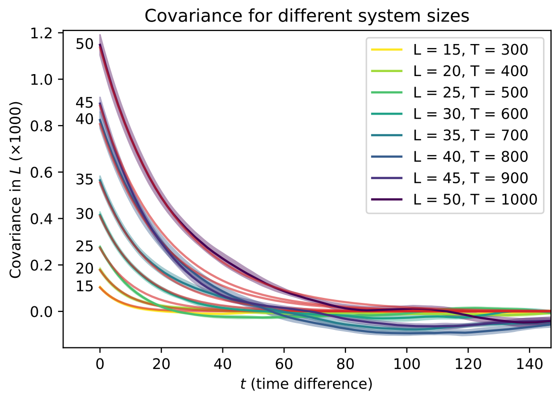


Figure 12: Estimates of the length covariance, labelled by L and show in units of 1000.

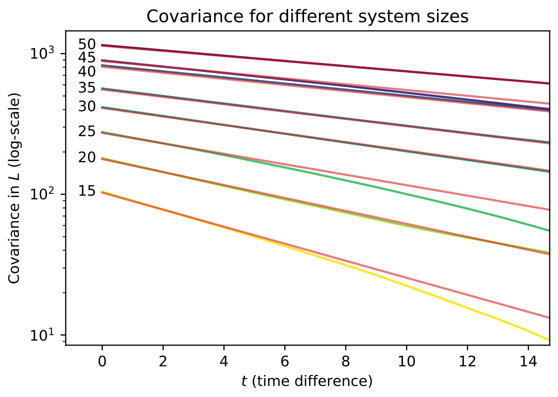


Figure 13: Log plot of the length covariance, labelled by L and including theoretical covariance.

Now, to compare these measurement results to theory it is important to realise that Eq. (15) is only valid for[13] $t \ll T$, which makes it difficult to fit a general exponential through the measured covariance as this exponential relation is only expected to hold for small t. An alternative is to visualise the theoretical covariance alongside the measured covariance such that we can at least visually inspect whether the data matches the expected results. To plot the theoretical covariance (15), a value for Λ is required. The obvious choice is to let $\Lambda = 1/L^2$ as is theoretically expected. However, we know from the measurements of σ_ℓ that it is underestimated. And since σ_ℓ^2 is precisely the normalisation of the covariance, using $\Lambda = 1/L^2$ would mean we have this underestimation for all t. And because we already analysed σ_ℓ, the absolute magnitude of the covariance is not that interesting; it is much more interesting to look whether the exponent of the covariance matches theoretical expectations. For this reason, we use the values of Λ estimated for the different L as seen in Fig. 11, such that we can check whether the rate of decay relates to the respective estimate of Λ in the correct way. The theoretical covariance with these Λ are also plotted in Fig. 12 in red, but are not very clearly visible. So to see the relevant behaviour at small t better, Fig. 13 shows the same data for small t with a logarithmic y-axis.

From Fig. 13 it appears that for small t the measured covariance is indeed linear as expected for the logarithmic plot. And comparing it to the theoretical covariance plotted in red, it seems that the measured covariance also shows the correct exponent, represented by the slope in the logarithmic plot. For larger t there is deviation from the theoretical line, which it to be expected as the theoretical equation has higher order corrections for larger t.

Another way to visualise how well the covariance for different L fit the theoretically expected covariance (15) is to scale the covariance and the time difference t such that all covariance measurements should collapse around one curve. To do this we scale the covariance by 2Λ and t by $1/2\sqrt{\Lambda}$, such that all curves should collapse around $\exp(-t)$. The results of scaling are displayed in Fig. 14 as well as a logarithmic plot for small t in Fig. 15, along with $\exp(-t)$ in red. From these plots it is evident that the curves collapse well for small t, and that there are discrepancies for larger t as expected.

[13] Again note that here T is the amount of timeslices and t the discrete time difference corresponding to the continuous time difference T in Eq. (15)

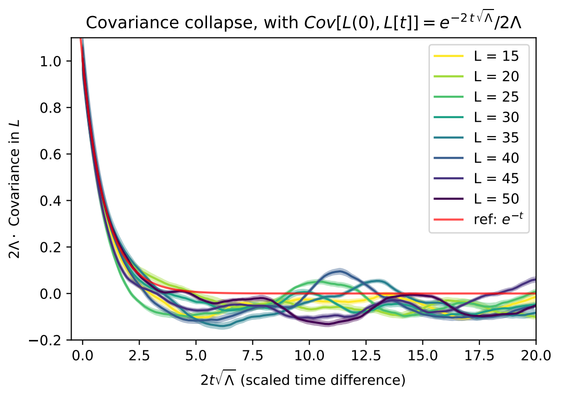


Figure 14: Scaled covariance with reference collapse function in red.

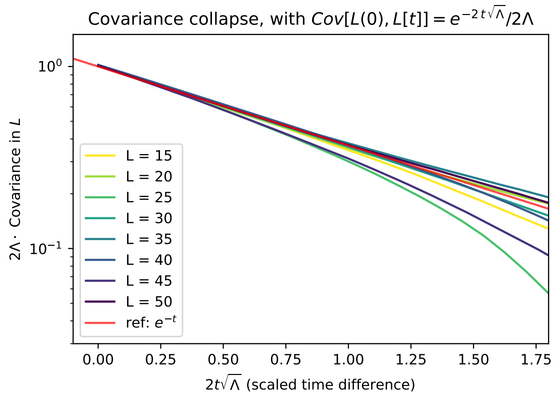


Figure 15: Logarithmic plot of scaled covariance.

3.3 Discussion

The determination of the equilibration time and correlation time is important to performing the simulation. However, for this project great effort was taken to determine these times to reasonable accuracy to be able to predict t_{eq} and t_{cor} for any given T and L. Knowing the computational effort that went into determining these still rather inaccurate predictions, doing it this way is likely not worth it; it is much more fruitful to pick the system sizes one wishes to simulate carefully and determine the equilibration and correlation time for these systems specifically. Unfortunately, we only later in the project realised that the interesting system sizes were $L \ll T$, so we could not optimally use the determined t_{eq} and t_{cor} anyway. Experience showed that taking the determined $t_{\text{eq}} = 200$ sweeps as burn-in time is also sufficient in this regime, and that the correlation time still scales with $N^{3/2}$ for a constant T/L ratio, however the crude approximation of t_{cor} does not seem to be accurate at higher T/L ratios. Also, although we expect t_{eq} to be independent of T, it should be checked to really make sure any system one does measurements on is thermalised, for if this is not the case and the total amount of measurements is low, the first few measurements could contribute to a significant error in the end results.

Moreover, the measurements of the standard deviation σ_ℓ and of the covariance $\rho_\ell(t)$ agree with the theoretical predictions within error margins. However, the found results appear have a systematic underestimation of the standard deviation and thus also the covariance. Since the theoretical result falls within the error, this may simply be coincidence and would require more measurements to decrease the error. However, this is likely a systematic error made due to having finite T, as $\ell(t)$ can only change a finite amount between adjacent timeslices, so having a finite T means that it is possible for $\ell(t)$ to fluctuate less from L than it could for $T \to \infty$. To investigate this, and in general the effect of different T/L rates, these measurements should be repeated for different values of T. But unfortunately we could not measure this within the time constraint of this project, as well as the scaling of computation time with $N^{3/2}$ making this computationally difficult.

Another point of this discussion is that the results of the covariance $\rho_\ell(t)$ measurements can only be checked visually, making it difficult to judge whether the results match the theoretical prediction. To make this more quantitative we would want to fit the expected covariance with a general exponential function and compare the different estimations of Λ as a result of this fit to each other and to $1/L^2$. However, to be able to fit an exponential we need a large enough part of the covariance $\rho_\ell(t)$ for which t can be considered ‘small’. This means that T needs to be larger for a given L than we currently use. It may also be possible to include higher order terms in the theoretical prediction of the covariance such that the theoretical prediction is valid for larger t.

Finally, since what we really want is to find the exponent of Eq. (15) for small t and this is equal to the slope at $t = 0$, we might be able to get a reasonable approximation of this slope by considering only the first few points and for example using spline interpolation or a polynomial fit to find the slope at $t = 0$. So we would again want to investigate this by using multiple T/L rates.

4 Conclusion

Using a Markov chain, we have been able to efficiently sample 2D CDT universes with a fixed volume. On these universes we were able to measure the length profile $\ell(t)$. With this length profile we computed the standard deviation σ_ℓ and the autocovariance $\rho_\ell(t)$. Using Monte Carlo methods, these were compared to a continuum theory for various system sizes at constant T/L ratio.

Both σ_ℓ and $\rho_\ell(t)$ seemed to fit the theoretical prediction quite well, though not perfectly. There are a couple of possible sources of errors. Perhaps most importantly, there are still significant statistical errors, mainly visible in Fig. 14. Furthermore, the theoretical model we used is only valid in the limit $T/L \to \infty$. It is of course not possible to reach this limit in practice. One possible future approach could be to further develop the theory to include higher order corrections. Another option is to investigate how the simulation results scale for different T/L ratios; so far we have only looked at the case $T/L = 20$. However, both of these approaches will likely first require a decrease in statistical errors.

The model we have used is quite efficient, and we have been able to extract some relevant data from it. However, we already have some suggestions that could (somewhat) improve the efficiency of the algorithm. Internally labelling the order four vertices differently would require no updates to the order four list when performing a shard move, and would thus allow for a decrease in computation time. It would also be interesting to investigate whether there is a significant difference in computation time between the two moves. If this is the case, it might be worth it to tune the move rate accordingly, since the correlation time seems to be relatively insensitive to small changes in the move rate. Besides these remarks on implementation, there is still a lot more information that can be extracted from this model; some suggestions have already been discussed in section 2.4. Furthermore, this model can be used as a stepping stone to investigating higher dimensional models, though this is certainly far from trivial.

References

[1] J. Ambjørn and R. Loll. "Non-perturbative Lorentzian quantum gravity, causality and topology change". In: *Nuclear Physics B* 536.1-2 (Dec. 1998), pp. 407–434. ISSN: 0550-3213. DOI: 10.1016/s0550-3213(98)00692-0. URL: http://dx.doi.org/10.1016/S0550-3213(98)00692-0.

[2] J. Ambjørn et al. "Nonperturbative quantum gravity". In: *Physics Reports* 519.4-5 (Oct. 2012), pp. 127–210. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2012.03.007. URL: http://dx.doi.org/10.1016/j.physrep.2012.03.007.

[3] J. Brunekreef and R. Loll. *Quantum Flatness in Two-Dimensional CDT Quantum Gravity.* 2021. arXiv: 2110.11100 [hep-th].

A Individual Contributions

For the project we divided the work as follows:

- *Literature research*: Is done roughly equally

- *Model design and programming*: Tom started with a vertex-based model, Jesse started a triangle-based model, and we optimized that together. Tom also started working on a shard model, but this was quickly discarded. Tom designed the CLI, and Jesse the visualisation.
- *Data analysis*: Jesse performed most of the data analysis, and collected all the data for the preliminary analysis. Tom collected the data for the main measurements and verified the validity of the data analysis.
- *Report writing*: Writing is divided roughly equally, generally Tom was responsible for the Introduction and Methods, and Jesse for the Results with Discussion.

For a very detailed overview of the individual contributions, they can be found in the commits of the GitHub repository, see appendix B

B Code and Data distribution

For this project we used a public GitHub repository (see `https://github.com/SirBlueRabbit/monte-carlo-CDT`) to share all project files and all relevant data. For an overview of the contents of the repository see the README.