S1336 - Project 3

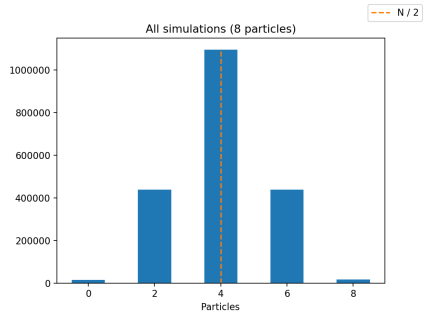
Erik Weilow

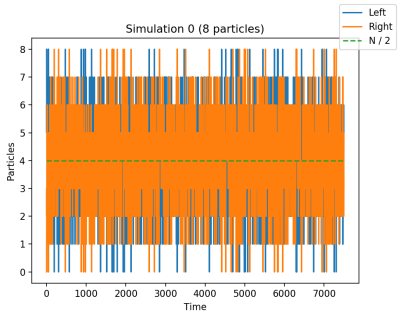
November 19, 2018

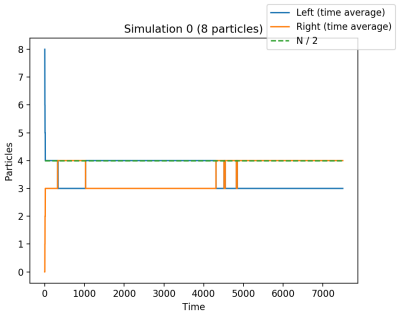
3.1

We'll first look at simulations for different N.

8 particles



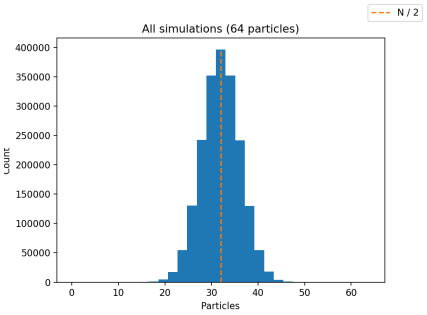


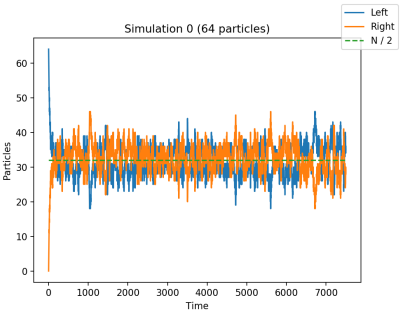


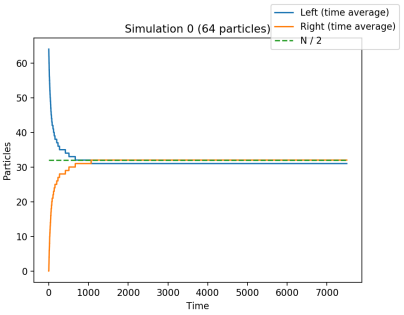
It's interesting that the histogram of simulation end values shows no simulations ending up in an odd count of particles in the left half of the box, when running an even number of simulation steps. This can be easily explained: since we start with an even number of particles in the left half, every subsequent step will first reduce or increase the particle count to an odd count. Every further subsequent step will reduce or increase the count to an even value, such that for every two steps the particle count either increases by 2, stays the same or reduces by 2. If the starting count is even, every 2 steps will also see an even amount of particles.

The simulation looks chaotic, but the time averages show that the particle count will be fairly split between left and right after some time.

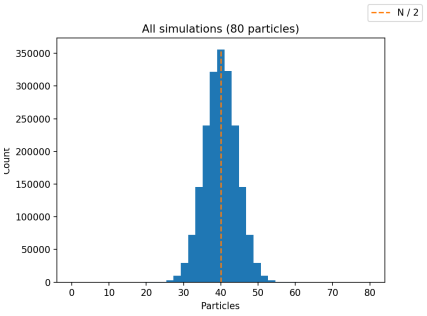
64 particles

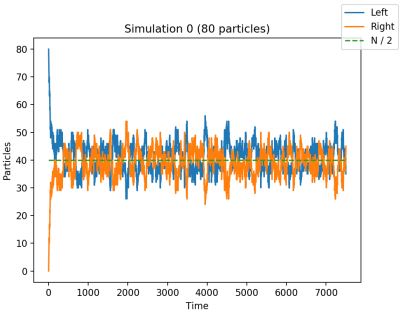


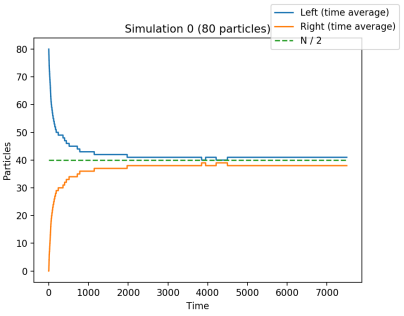




80 particles





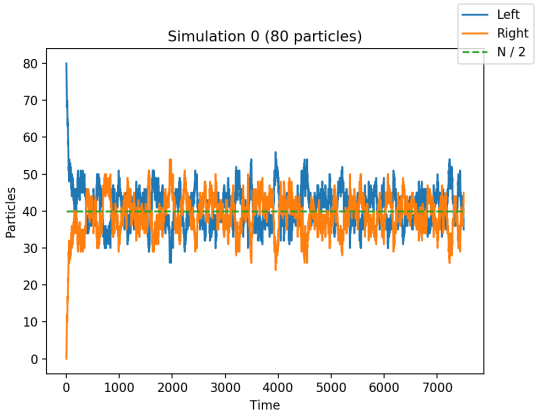


The behaviour of the whole system becomes much more clear with an increased amount of particles. It's clear that the time average of the system tends towards an even split of particles between the left and right sides of the box. This is shown both by the histogram showing a mean of half the particle count, as well as the time average for left and right sides trending towards each other for large time.

Equilibrium

If we define equilibrium as a constant time average, as can be shown in the previous plots, it's clear that the systems on average reach a statistical equilibrium. This must be taken with a bit of caution however, as none of the simulations are in a true equilibrium as the particles in each side randomly fluctuate around the mean.

This can be seen in the following simulation:

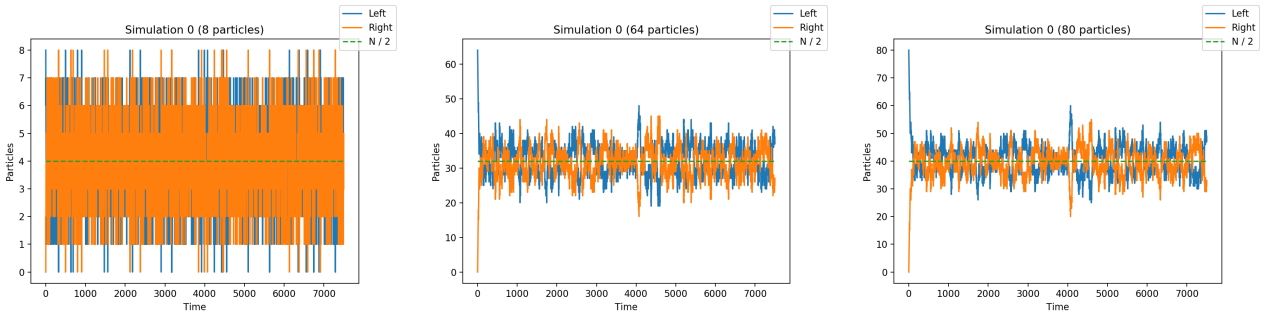


3.2

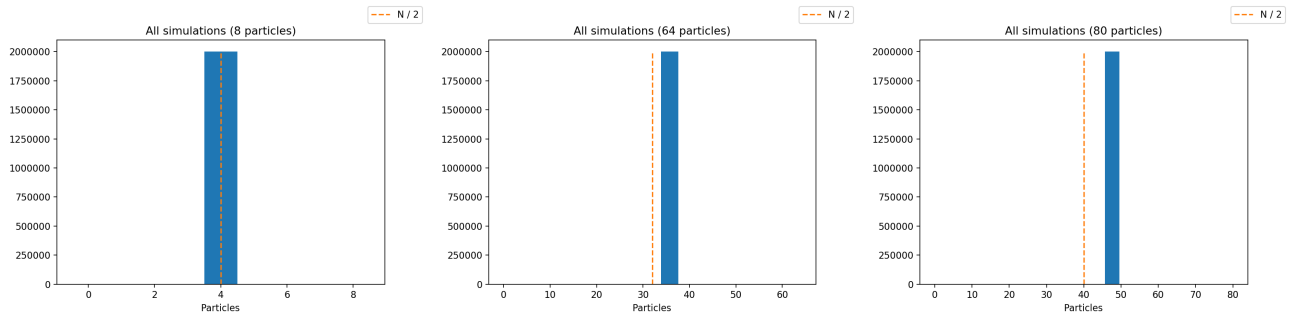
The qualitative behaviour of $n(t)$ is that it statistically tends to $\frac{N}{2}$ as $t \to \infty$, as shown in the previous section.

3.3

The effects of seeding the random number generator isn't clear at a glance when looking at just a single simulation (for different N):



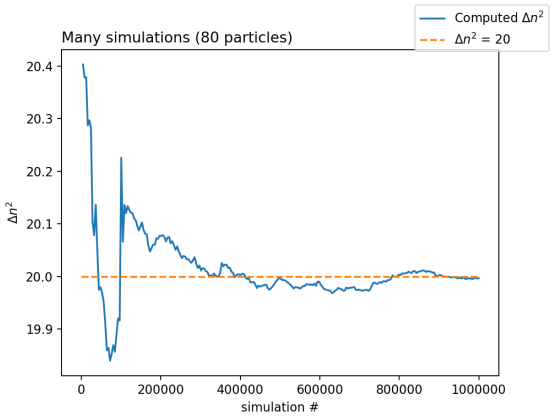
However, if we look at the histogram of simulation end values the picture becomes clear:



It can be concluded that seeding the random number generator has the effect of producing the exact same sequence of random values for every simulation, thus making every simulation exactly same.

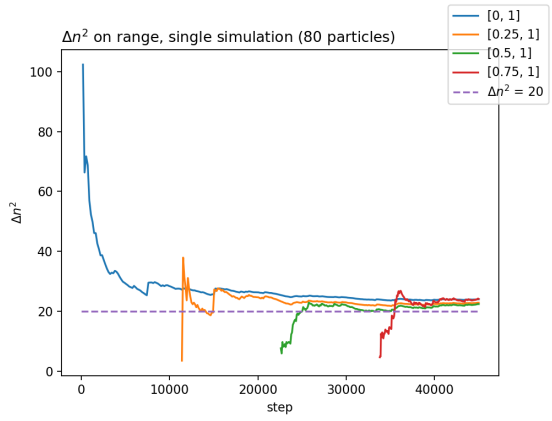
3.4

We'll start by looking at Δn^2 for many simulations:



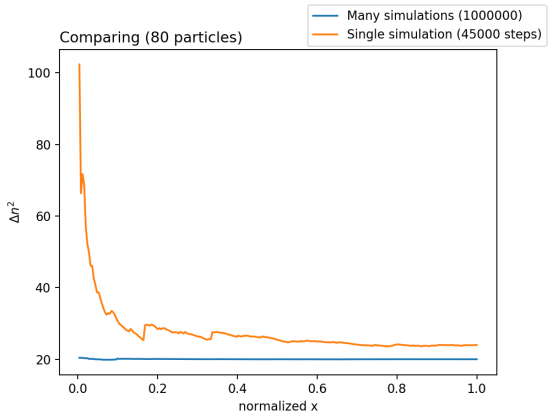
The value comes very close to 20 after calculating Δn^2 for a million simulations of 7500 each.

This can be compared to calculating the same value for different parts of a single simulation with 45000 steps:



The value here tends slightly higher than 20.

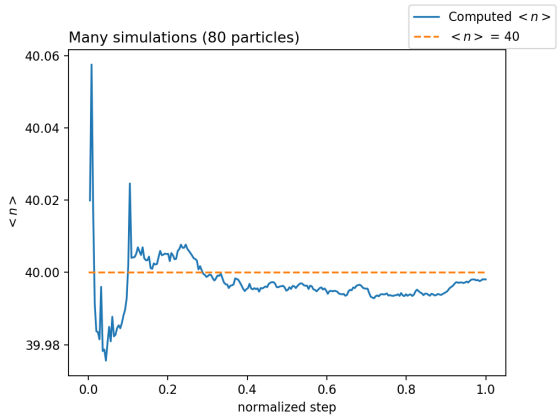
When placed in a single plot, the values of Δn^2 for many simulations can be compared to a single simulation:



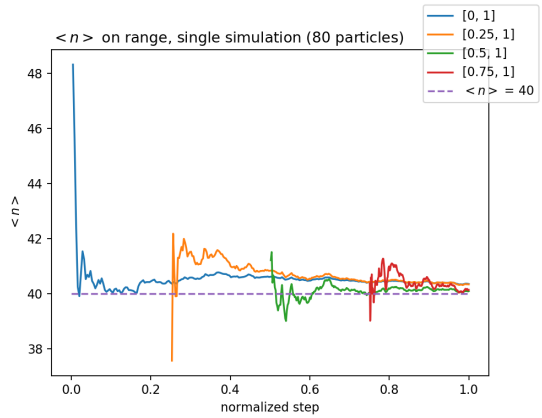
The value given by computing many simulations have converged well on 20, but the value given by the single simulation is not as close to converging on a value.

3.5

We've already seen in section 3.1 that it is expected for the number of particles in both half to statistically average $\frac{N}{2}$.

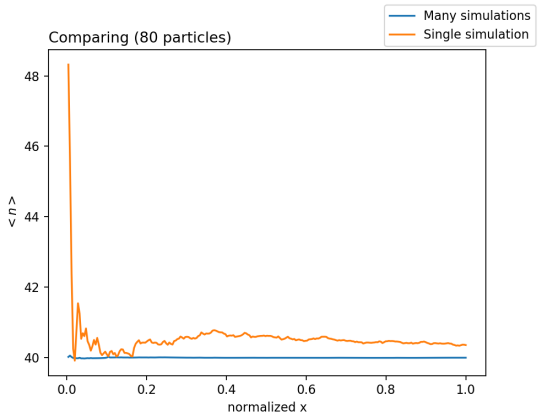


When computing the average $< n >$ from the end values of many simulations, the result tends close to what is expected.



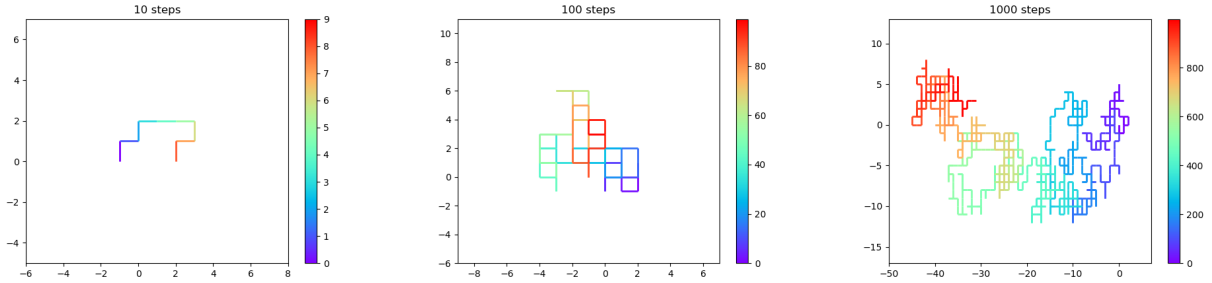
When computing on different parts of a single simulation, the result is much further from what is expected.

These conclusions can be summarized in a single plot:



3.6

The first thing to do is look at some sample walks with lengths 10, 100 and 1000:



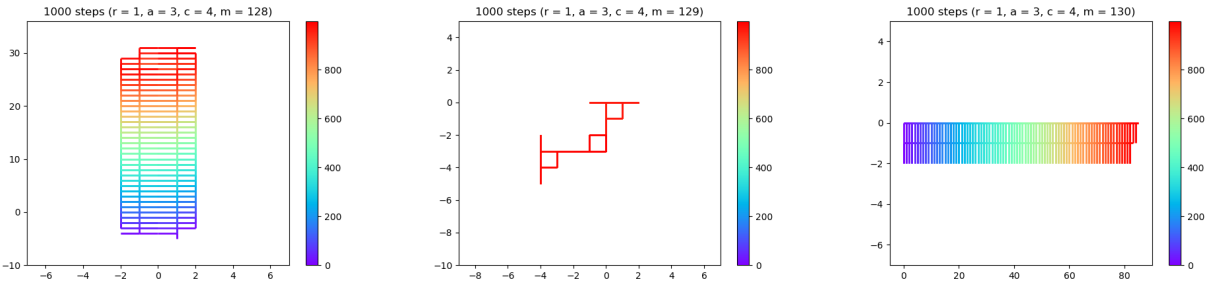
The walks are colored from blue to red, according to the rainbow color scales, to easily show where they start and where they end.

3.7

By introducing the custom random number generator $r_n = (ar_{n-1} + c)\%m$ and setting

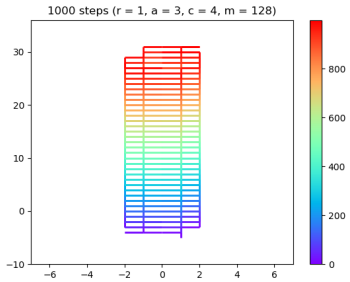
$$r_0 = 1, \quad a = 3, \quad c = 4, \quad m = 128, 129, 130$$

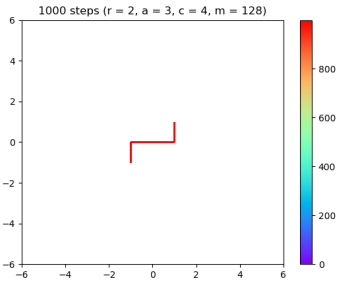
we can generate the following (deterministic) walks:

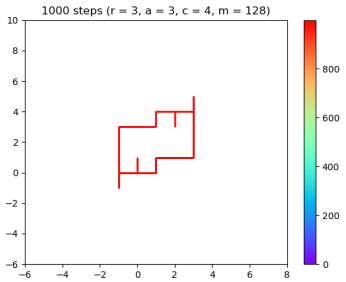


Other parameters can be varied, generating the following walks:

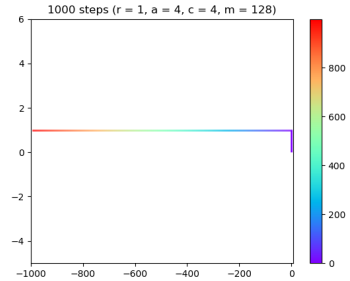
Varying r_0

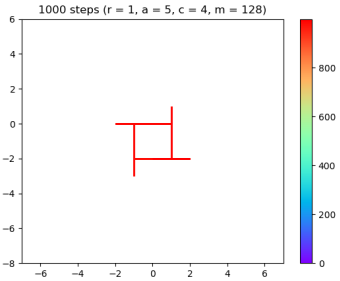


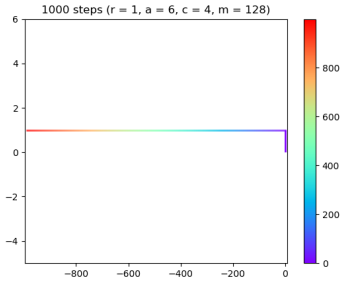




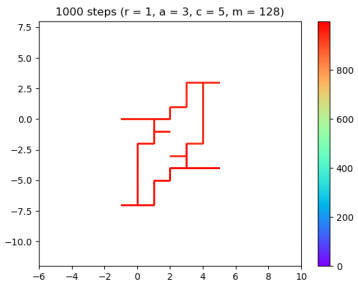
Varying a

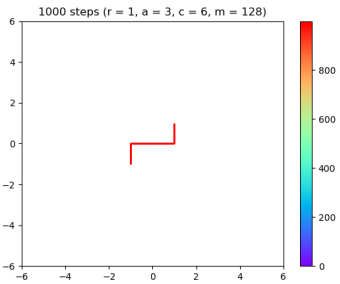


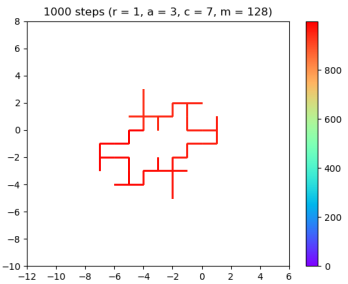




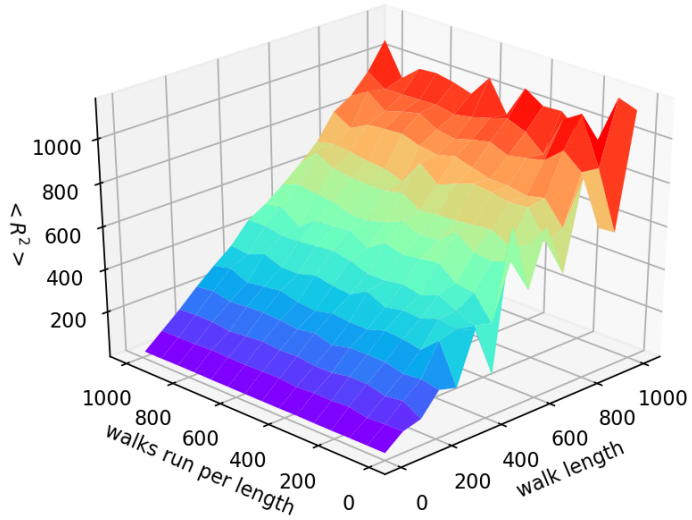
Varying c







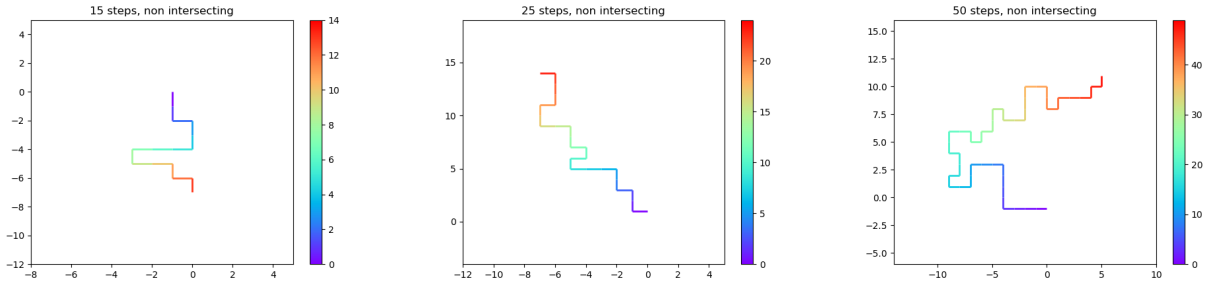
3.8



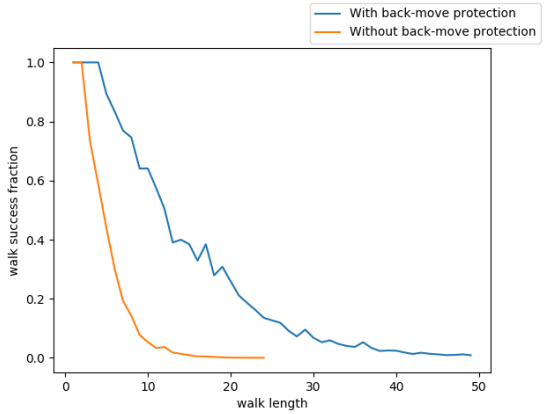
I found this way of visualizing walk length against $< R^2 >$ to be quite effective. It captures both walk length against $< R^2 >$, but also how $< R^2 >$ tends with increased number of walks.

3.9

When we have such stringent requirements of a non-self-intersecting random walk, it must be verified that the algorithm works:

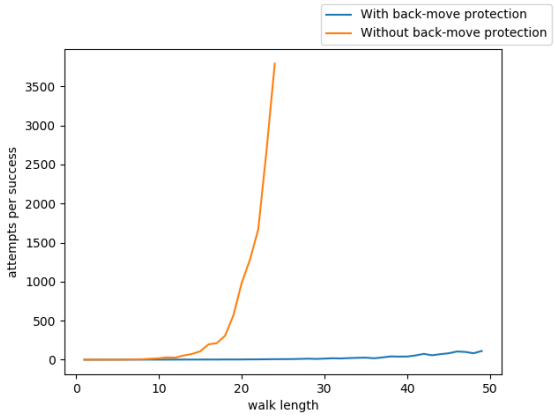


Looking at the success fraction, it's clear that the method of not going back the same direction as the direction each step came from improves a bit:



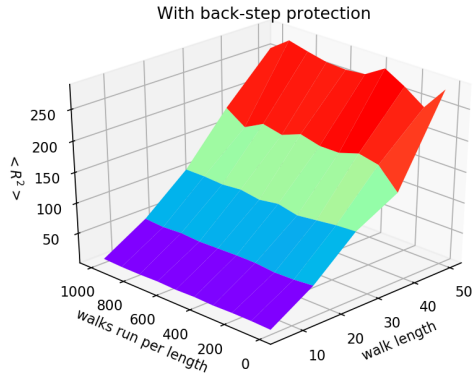
Instead of tending to 0 around a walk length of 20, more than double the walk length can be done before reaching the same low success fraction.

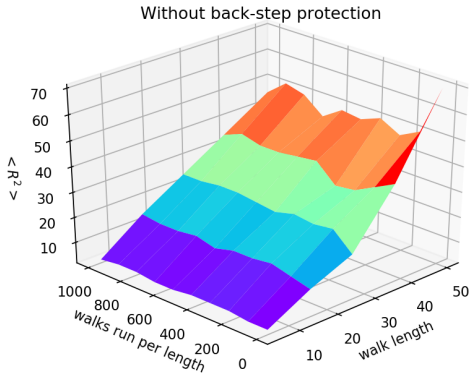
I feel that this plot is a bad way of showing just how much better the back-move protection is. Instead we can look at the average number of self-interesecting walks per non-self-intersecting walk:



The pure random walk basically becomes insanely hard to find solutions for with a walk length of much more than 20-30, but the method where the walk doesn't return in the same direction is *MUCH* better.

3.10





These plots share the same structure as the plot in section 3.8. It's clear that back-step protection (the right plot) sees $< R^2 >$ only a fifth of the walks without back-step protection.