# Lab 10: PHY5341 Fall 2022

David J. Gayowsky #8544603
sgayo008@uottawa.ca

November 21st 2022

# Question 1: Travelling Salesperson

**1. a. Write a simulated annealing code based on the Metropolis algorithm to solve the travelling salesperson problem for $N$ cities chosen by your code in a random location within a unit square. Test your code first for a small number of cities, so that you can judge your results by hand.**

In order to keep track (and avoid confusion), we will consider the following algorithm with which to compute our travelling salesman problem:

1. Generate a random configuration of $N$ cities on an $M$x$M$ lattice, where each city $n_i$ is given some coordinate $(m_i, m_j)$, and where we assume the lattice is at some initial temperature $T_0$. As these cities are already of randomized coordinates, we will consider this to be the first valid route.

2. Generate a trial configuration of the order in which we visit cities by swapping the order of two of the indices $n_i$ and $n_j$ in the current valid route.

   (E.g. for some valid configuration one may swap two indices such that $\{n_a, n_b, n_c, n_d\} \longrightarrow \{n_a, n_d, n_c, n_b\}$.)

3. Compute the change in route length $\Delta d$ ("energy") given the new trial configuration, where the change in route length will be:

   $$[d(n_{i-1} \to n_j \to n_{i+1}) + d(n_{j-1} \to n_i \to n_{j+1})] - [d(n_{i-1} \to n_i \to n_{i+1}) + d(n_{j-1} \to n_j \to n_{j+1})]$$

   Where $d$ represents the distance computed between the given points using simple Pythagorean theorem (and where we assume we may travel diagonally across the lattice).

4. If $\Delta d$ is $\leq 0$, i.e. the new route is shorter, accept the trial configuration as the next valid route.

5. Otherwise, if $\Delta d > 0$, i.e. the new route is longer, compute the probability of the new route at the current temperature $T$, given by:

   $$P = e^{-\frac{\Delta d}{T}}$$

   And generate a number $u$ from the uniform random distribution $U(0, 1)$. If $u \leq P$, keep the trial configuration. Otherwise, discard it and return to the last accepted valid route.

6. Calculate the mean route length for the last `mean_samples` calculated route lengths, and compare it to the mean route length for the last `2*mean_samples` to `mean_samples` calculated route lengths. If the mean route length has decreased by less than some predetermined tolerance `roc_tol_at_T`, say that we have reached a minimum route length for this $T$. Otherwise, repeat steps 2-6.

7. Calculate the mean route length for all routes generated at $T$.

8. Repeat steps 2-7, varying $T$ by $\Delta T$.

9. If the mean route length has decreased by less than some predetermined tolerance `roc_tol` after varying $T$, or if $T$ reaches $\sim 0$, say that we have reached a minimum route length, and that we have reached a global minimum route length for all $T$. Otherwise, repeat steps 2-7 and 9.

Here, this generalized algorithm is used to solve the travelling salesman problem, and is contained in functions included with assignment code submission. Calling function `travelling_salesman` will generate plots as required.

In addition, for small $N$ where there are only 1-12 possible routes, it is assumed that at some point we look at all possible route lengths. Thus, we create a set of possible route lengths which is returned with the code, and ensure our code works for small $N$ by ensuring our code picks the smallest. (Note this is only done to three decimal places, as due to small amounts of error that accumulate while calculating lengths, not all lengths are exact to floating-point precision.)

Thus, we generate figures for $N \in \{3, 4, 5, 6\}$, and evaluate their results as shown in Fig. 1-4 below.

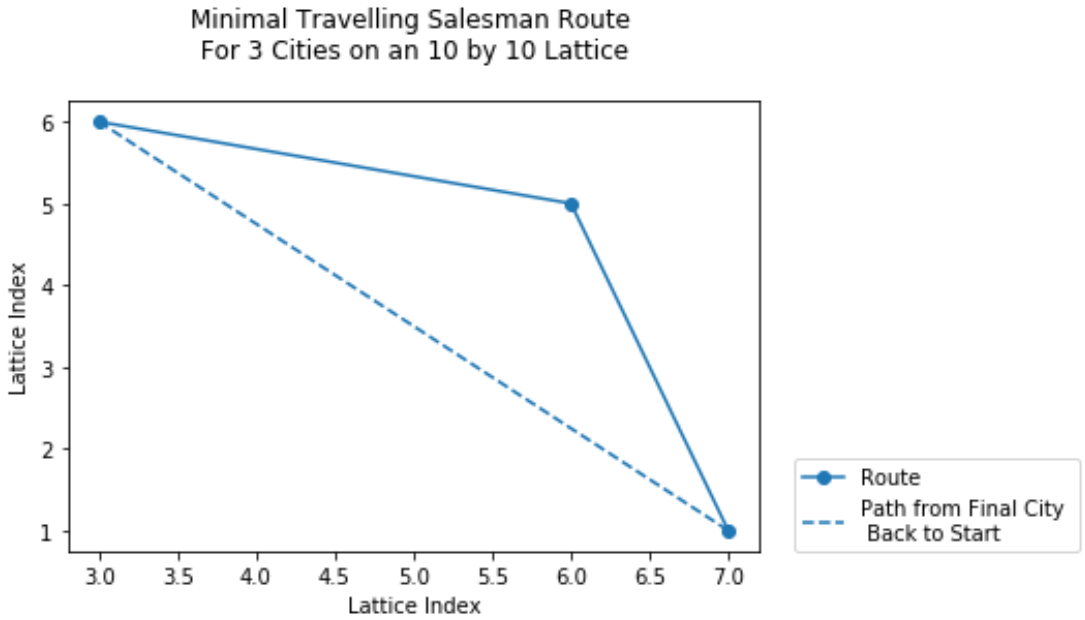



Figure 1: Minimal travelling salesman route for 3 cities.

Where the set of route lengths and minimized route for $N = 3$ is:

```
Set of Unique Route Lengths: {13.689}
Final Route Length: 13.688507523218892
```

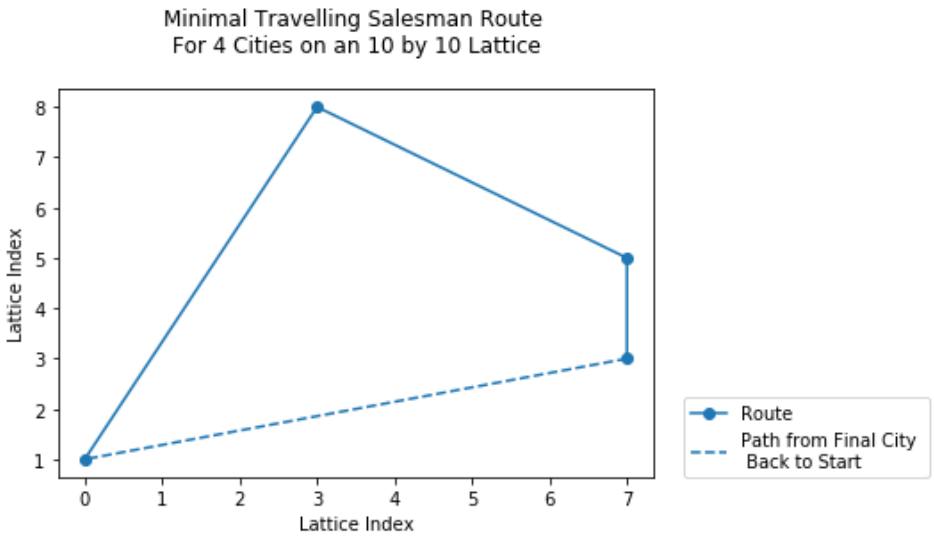



Figure 2: Minimal travelling salesman route for 4 cities.

Where the set of route lengths and minimized route for $N = 4$ is:

```
Set of Unique Route Lengths: {21.896, 24.081, 26.745}
Final Route Length: 21.895882995144426
```

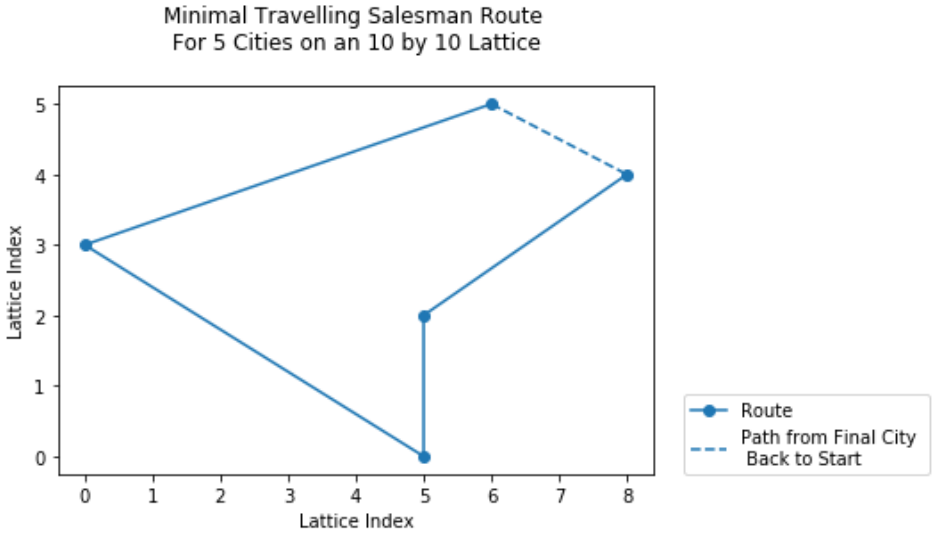



Figure 3: Minimal travelling salesman route for 5 cities.

Where the set of route lengths and minimized route for $N = 5$ is:

```
Set of Unique Route Lengths: {25.76, 19.997, 20.66, 21.328, 21.292, 23.923, 24.549,
    25.091, 26.423, 22.496, 21.871, 25.128}
Final Route Length: 19.997126468145836
```

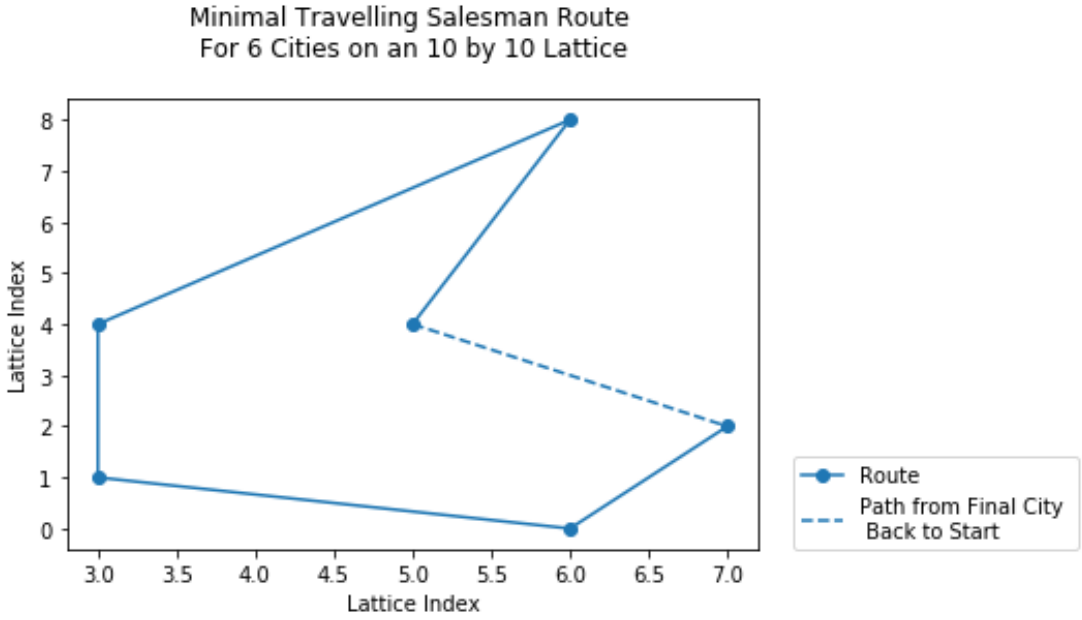



Figure 4: Minimal travelling salesman route for 6 cities.

Where the set of route lengths and minimized route for $N = 6$ is:

```
Set of Unique Route Lengths: {26.286, 25.058, 27.93, 27.935, 27.075, 27.053, 27.329,
    27.202, 28.69, 29.457, 20.604, 20.35, 22.605, 23.609, 24.237, 25.57, 26.452, 26.446,
    26.689, 27.323, 27.068, 28.811, 28.65, 29.567, 29.324, 26.54, 27.456, 28.517, 29.284,
     30.039, 31.522, 24.047, 25.073, 24.963, 24.088, 25.679, 25.114, 25.881, 24.67,
    25.004, 24.197, 25.098, 25.437, 24.803, 25.68, 30.334, 22.599, 22.087, 22.843,
    23.482, 30.899, 24.491, 24.965, 25.586}
Final Route Length: 20.349878388032018
```

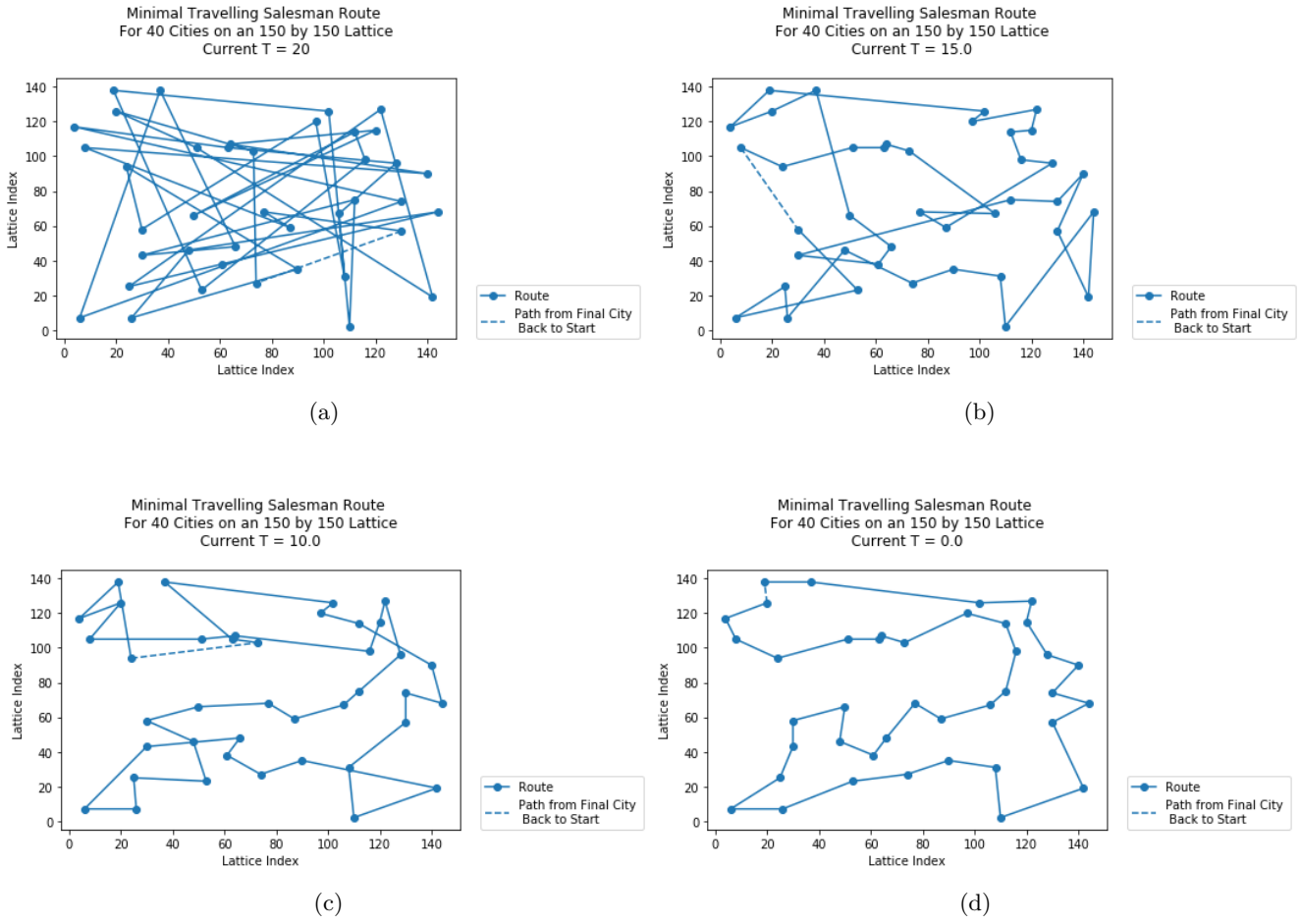
Thus, we can see here that this algorithm works (insofar as we know!) for small $N$.

However, we do include a catch here - for small $N$, to double check our answer, we include an optional `failsafe` variable in our `travelling_salesman` function. Here, we compare our "final" route length to the set of unique route lengths generated by our algorithm, and we only accept this answer if it is equal to or less than the smallest element in our list of unique route lengths. Otherwise, we restart the algorithm and hope we won't get stuck again.

Note that we can't use this for large $N$ - the whole point is to not have to calculate all possible route lengths! But it is a nice little failsafe to double check that our algorithm is doing the right thing here, which we see it is.

**1. b. and c. Perform your simulation for larger numbers of cities. Plot the evolution of your route, similar to Fig. B.4 from Giordano, shown in class, and include your minimizing route.**

Here, we once again use our `travelling_salesman` function to plot the evolution of these routes for $N = 40$ cities on a 150x150 lattice, and $N = 80$ cities on a 500x500 lattice, as shown in Fig. 5 and 6 below.




(a) (b) (c) (d)

Figure 5: Evolution of shortest route to traverse $N = 40$ cities on a 150x150 lattice, at (a) $T = 20$ (initial route), (b) $T = 15$, (c) $T = 10$, and (d) $T = 0$ (final route).

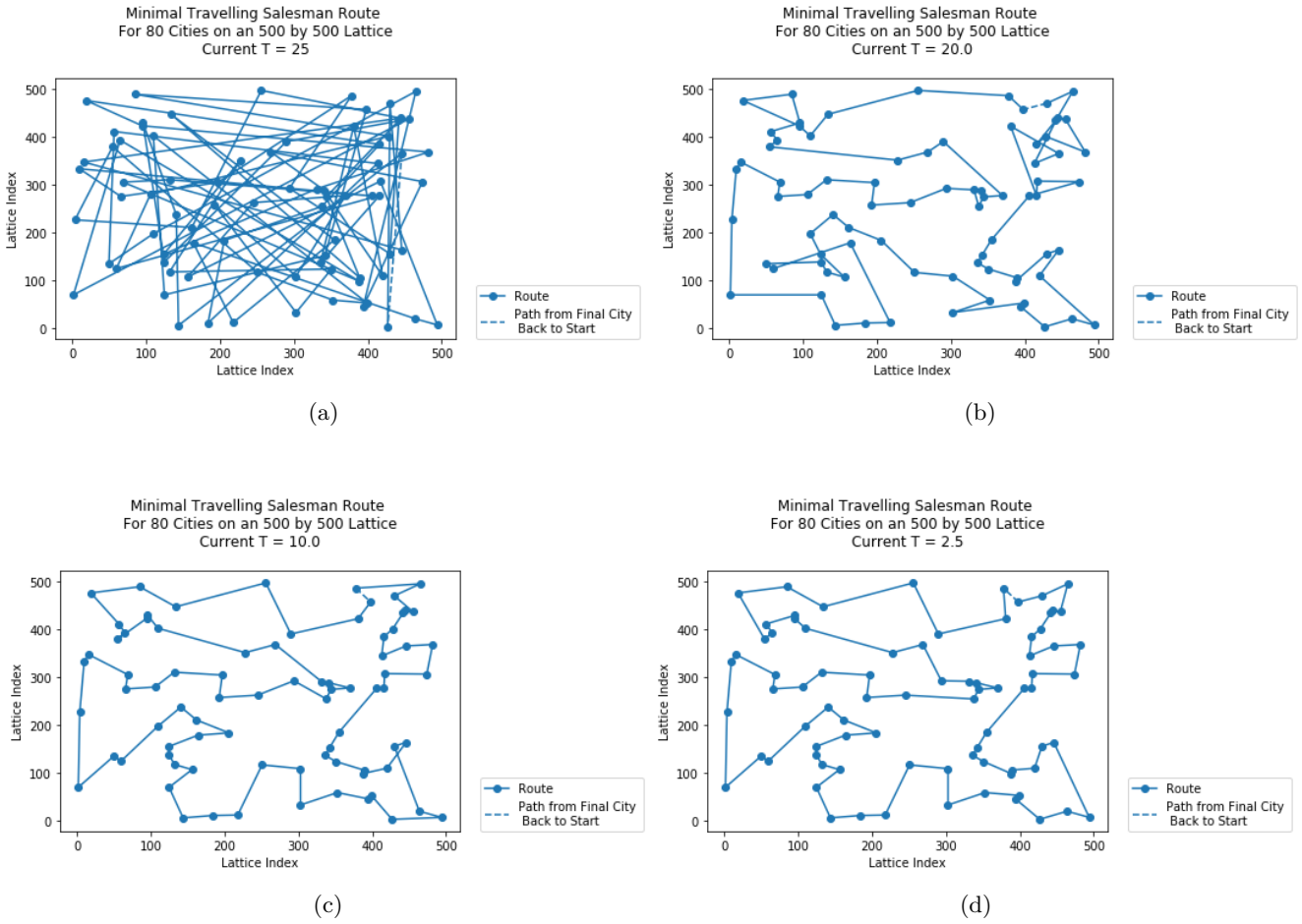



Figure 6: Evolution of shortest route to traverse $N = 80$ cities on a 500x500 lattice, at (a) $T = 25$ (initial route), (b) $T = 20$, (c) $T = 10$, and (d) $T = 2.5$ (final route).

**1. d. Briefly describe your computational approach (including how you set up your problem, your annealing schedule, your initial "temperature", how you determined equilibrium was reached at each temperature, how you decided when the minimizing route was reached, and any other relevant computational aspects.)**

Here, the following considerations were taken (in addition to the generalized algorithm described in part a.):

- The problem was set up with a series of two nested functions (while loops) - one which would increment the temperature downwards, and one which would equilibriate the function at a given temperature $T$.
- Here, the annealing schedule varied by $N$. For small $N$ where we can demand higher accuracy of results without having a prohibitive computational time, the annealing schedule varied $T$ by reducing by 0.05. However, for large $N$, we considered where $T$ would vary by 0.5 in order to reduce overall computational time.

One note here is that as larger variations occur at higher temperatures, one could scale the amount that $T$ varies as we decrease through our annealing schedule - although here, due to our choice of measuring convergence at a given temperature, we do not consider it strictly necessary in this case to scale $\Delta T$ based on $T$. In other algorithms, however, this may be a necessary step.

- The initial temperature was also dependant on $N$, where a larger $N$ necessitated a higher initial temperature. In the case of $N \leq 4$, an initial temperature of 5 was sufficient, whereas in the case of $N \in \{6, 40\}$, an initial temperature of 20 was needed, and in the case of $N = 80$, an initial temperature of 25 was needed. In all cases, the temperature directly defines how "far" our algorithm can step outside of the current route - a larger temperature allows greater variation in the route. Thus, for larger $N$, a larger initial temperature is needed to ensure that we cover the full range of possible routes, and allow our algorithm to fall into the global minima, rather than getting trapped in some arbitrary local minima.

- At each temperature, equilibrium was reached by comparing the means of the last `mean_samples` route lengths with the previous `mean_samples`, and seeing whether the mean changes by some input `roc_tol_at_T` amount. If the mean of the route length remains stable over these two sets of samples, we consider that we have converged at temperature. Of course, varying the value of `mean_samples` and `roc_tol_at_T` is a balancing act - too small tolerance and your runtime becomes prohibitive, too small a number of `mean_samples` and you run the risk of finding a zero change for the incorrect value, but too large and you may once again find the incorrect value by virtue of converging to the true mean as the number of samples grows large.

- In the small $N$ case, it was simple enough to enact our failsafe to make sure our minimizing route was reached - however, for large $N$, the whole point is not to have to do this!

  Thus, for large $N$, we have two possible methods: one, we consider once again a change in the mean route length, such that if the mean route length changes by less than some predetermined `roc_tol`, we have converged.

  Secondly, if `roc_tol` is not reached, we analyse our routes graphically once our annealer reaches $T = 0$, rather than computationally. We assume from Giordano et al, and our own knowledge of geometry, that in general the *outer* path will be the shortest (or sufficiently near the shortest). We consider that the shorter path will not cross over the interior of the given polygon - and so, we say that if we have a path segment which does so, this is likely not our shortest path. Similarly, if our path is entirely exterior, i.e. there are no crossing paths, we consider it is *likely* (although not *guaranteed*) to be the shortest path.

### 1. e. Briefly discuss your results.

Here, we note that:

- At small $N$, i.e. for $N \in \{3, 4, 5, 6\}$, correct results were achieved by taking small `roc` tolerances, variances in temperature, and a lower initial temperature. These results were explicitly analysed to ensure that the correct result was found (which it was!).
- At larger $N$, i.e. for $N \in \{40, 80\}$, it was considered that the "correct" result was achieved when the path was no longer overlapping. In both cases, this path was found, however larger variances in temperature, `roc` tolerances, and a higher initial temperature were required in order to ensure that the algorithm would not get stuck in a local minima.
- It was noted that in our large $N$ cases, both of our possible "minimal route" evaluation criteria were used - in the $N = 40$ case, we did not reach a sufficient `roc_tol`, and thus evaluated our path graphically at $T = 0$, which we found to be sufficient (as we have no path crossings).

  However, in the $N = 80$ case, we noted that our algorithm converged and reached a mean change in path length less than `roc_tol` at $T = 2.5$ - thus, we can say that we consider this to mathematically be the smallest possible path length. Analysing this path graphically, we echo this result, considering that again, we observe no path crossings, and consider this to be the smallest possible path within our range of accuracy.
- Additionally in both large $N$ cases, the algorithm behaves as does the algorithm and path evolution plots given in Giordano, reassuring us that our algorithm was implemented correctly.
- It should be noted that it is still possible for our algorithm to fall into a local minima - as we may have well done for our large $N$ cases - however, the use of a properly large initial temperature, allowing us to step across the full range of possible path lengths, mitigates this risk, thus we may consider our results "correct" within this assumption.

# Question 2: Genetic Algorithm

**2. a. Describe how you are performing recombinations (from reproduction) and mutations.**

Here, to avoid confusion, the generalized algorithm for the Ising spin glass genetic algorithm is given by:

1. Generate an $N \times 2N$ random array of $J_{ij} \in \{-1, 1\}$ which serve as the nearest-neighbor interaction values, where we are modeling on an $N \times N$ lattice.
2. Generate an initial population of $M$ random $N \times N$ site ising configurations, such that each site has a spin of $\pm 1$.
3. Calculate the energy of configurations, using the array of $J_{ij}$.
4. For each set of two ising configurations in the set of configurations $\{M\}$, generate two "offspring" by comparison and generation of random spins by:
   (a) For each spin index, compare the two parent spins. If the two parent spins at that lattice site are the same, assign that spin to the offspring.
   (b) If the parent spins are different at a given spin index, randomly pick a spin $\in \{-1, 1\}$ to assign to that index in the offspring configuration.
5. For each "offspring" configuration, apply mutations by:
   (a) For each spin index, generate a random variable $u \in U(0, 1)$.
   (b) If $u \leq R$ for a given mutation rate $R \in U(0, 1)$, flip the spin at that site. It follows from this that a larger mutation rate causes an increase in offspring mutations, as we expect.
6. Calculate the energy of the offspring configurations.
7. Keep the two configurations with the lowest energy, replacing parent configurations with offspring if necessary, and storing energy values.
8. Calculate the number of times the most common energy occurrs in our population.
9. If the frequency of this most common energy is within some declared tolerance value `tol`, say that we have converged to the ground state, knowing that ideally all states will eventually converge to the ground state.
10. If we say we have converged, find the configuration with the lowest energy value and assume this is our "ground" state.

Here, the method of performing recombinations from reproduction and mutations are given in steps 4 and 5 of the algorithm.

(Now, we do note this isn't a *great* algorithm - unfortunately it involves calculating all energies rather than just energy differences, since I didn't want to get into the rigamaroll of having to manually compare both offspring with both parents and each other etc. - much easier to just have python find minimum values in each array.)

**2. b. Consider 4x4, 6x6 and 8x8 spin square lattices. Plot your found ground state for each lattice size.**

Here, note that the found ground state will depend on the random combination of $J_{ij}$'s generated during each algorithm run - thus, we do not expect to see exactly the same pattern in all cases. In addition, note that we strictly consider the $B = 0$ case, and thus do not consider this term in our calculations.

Thus, the required plots are shown below in Fig. 7, 8, and 9, respectively, where note the required fraction of states required to be of the same (minimal) energy was lowered for larger $N$ due to computational time requirements.

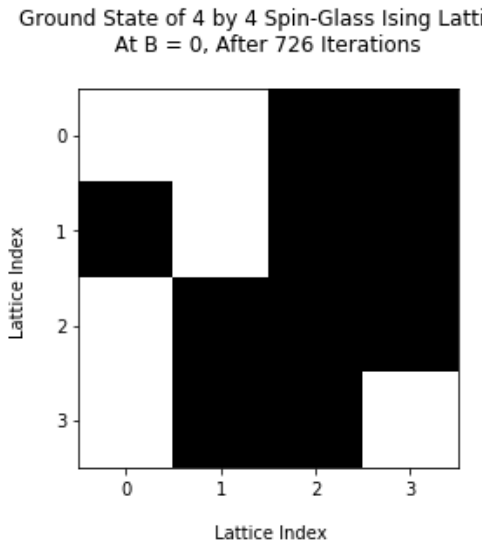



Figure 7: Ground state for 4x4 spin-glass Ising lattice, $B = 0$.

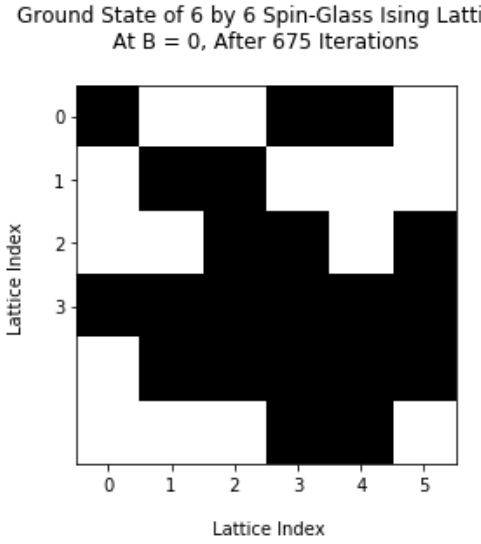



Figure 8: Ground state for 6x6 spin-glass Ising lattice, $B = 0$.

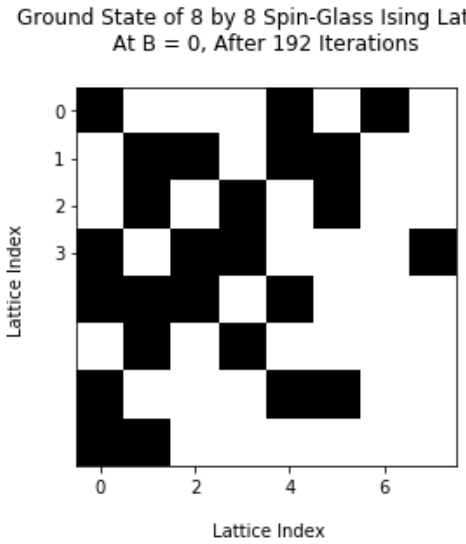



Figure 9: Ground state for 8x8 spin-glass Ising lattice, $B = 0$.

**2. c. What is the mean number of generations required to find the ground state as a function of lattice size? Are there any general trends?**

Here, the mean number of generations required to find the ground state depended heavily on:

- The tolerance (i.e. number of states at minimal energy required to declare a "ground state"), where a lower tolerance required a lower number of states at minimal energy.
- The allowed amount of mutations, where a larger amount of mutations would allow more variation of states, and possible faster convergence, however had a chance of destroying inherited "genetic" information.
- The lattice size, where a larger lattice size converged slower.

Overall, the mean number of generations required to find the ground state increased with lattice size, simply due to the amount of possible variations in configurations.

**2. d. How do your results in a) and b) change if you double the population size?**

As population size increases, so does the required number of iterations. However, an increased population size will also give a more accurate final result.