## 1. Change the number of cities in the final map (between 10 and 30). How are your results affected? Why?

The computation is slower if there are more cities.

Running, 50 simulations we get:

30 cities:

DescribeResult(nobs=50, minmax=(4192.1634079981523, 6660.5804125435261), mean=5628.1641217958959, variance=259257.33107768165, skewness=-0.32174959580050894, kurtosis=0.2053648701358579)

10 cities:

DescribeResult(nobs=50, minmax=(1426.7415970017191, 2409.3372546616974), mean=1859.2754770013023, variance=67610.280709010141, skewness=0.6135001753664528, kurtosis=-0.6989240782360522)

With less cities, we see a tighter distribution, so if we add more cities, the solution is less accurate. This is intuitive since we are increasing the complexity of the problem.

## 2. Change the alpha and temperature parameters. How do they affect the results?

For alpha, I run 50 simulations:

Alpha = 0.99, 10 cities

DescribeResult(nobs=50, minmax=(1426.7415970017191, 2120.9053515689425), mean=1693.4425431608827, variance=34511.175127528681, skewness=0.2734900652610194, kurtosis=-0.3367026253509846)

Alpha = 0.9, 10 cities

DescribeResult(nobs=50, minmax=(1434.515429558543, 2475.9795773111082), mean=1978.1013317861607, variance=84069.125591072036, skewness=0.22797736013142147, kurtosis=-1.1428934064742353)

If alpha is closer to 1 our solution has less variance and is more accurate. This is because the temperate decreases slower and therefore we tend to move around randomly for a longer period of time (since the probability stays closer to 1 for longer). We also do more iterations if the temperature deceases slower.

For the initial temperature again I consider two cases:

T = 1e2, 10 cities:

DescribeResult(nobs=50, minmax=(1426.7415970017194, 2484.1459315034208), mean=2021.3658458469995, variance=89962.282135042042, skewness=-0.3986291146719836, kurtosis=-1.0399826540449109)

T = 1e20, 10 cities:

DescribeResult(nobs=50, minmax=(1434.515429558543, 2475.9795773111082), mean=1978.1013317861607, variance=84069.125591072036, skewness=0.22797736013142147, kurtosis=-1.1428934064742353)

If the initial temperature starts larger there is more probability for moving around more at the beginning. Due to the exponential decay, the alpha parameter dominates the decreasing nature of the schedule function. The initial temperature barely matters. We should focus on choosing alpha.

## 3. Use a different schedule function (something other than exponential decay). Is the algorithm still effective?

I tried a linear decrease: T0 – alpha \* t. I found that it was hard to tune alpha and the initial temperature to even get the algorithm to converge. The temperature decreased too slowly.

Empirical research was done here:

<http://what-when-how.com/artificial-intelligence/a-comparison-of-cooling-schedules-for-simulated-annealing-artificial-intelligence/>

They show that simulated annealing works best when the curve has a moderate slope at the initial and central part of the process and smaller slope at the tail. This is exactly what the exponential function does. They find that the specific shape of the curve does not matter much.

## 4. Use a different successors function; e.g., generate successors of a state by swapping *any* pair of cities in the path, rather than only adjacent cities. Try defining your own successor function. What effect does the change have?

Check out page 64 of this article for the best successor function:

<http://www.stat.yale.edu/~pollard/Courses/251.spring2013/Handouts/Chang-MoreMC.pdf>

My attempt:

I tried using all permutations. So the number of successors grows at O(n!) instead of O(n). The problem becomes unwieldly very quickly if n > 7. With 5 cities, both successors’ methods give the same results. A successor method growing faster than linear time will be inefficient for problems with more than 10 cities. For example, exp(10)=22026 and 10!=3628800.

I also built a successor function that return n random permutations. So basically you don’t compare your neighbors, but instead compare random points on the surface. It is a bit more expensive than the regular successor function, but not too bad.

These are the results for the random permutations with 15 cities:

DescribeResult(nobs=100, minmax=(1375.4334412550093, 1968.1278032868058), mean=1759.7284399419909, variance=6330.5719051811102, skewness=-0.9347364977178277, kurtosis=4.36493097084393)

For the regular successor function, we get:

DescribeResult(nobs=100, minmax=(802.68295183093824, 1346.9780085984244), mean=1012.710915438476, variance=22963.525923545065, skewness=0.7302988117332708, kurtosis=-0.8227020778776559)

The spread of the random permutation successor function produces less variance.

## 5. Use a different distance metric for get\_value (e.g., we used the L2-norm (Euclidean distance), try the L1-norm (manhattan distance) or L∞∞-norm (uniform norm)

It seems that the manhattan distance is the most stable. But I don’t know why?

Results:

Euclidian simulation (sims=100, 10 cities)

DescribeResult(nobs=100, minmax=(1347.095088678999, 2119.7823210957645), mean=1563.4921557398256, variance=45207.292687141722, skewness=1.1025570603172503, kurtosis=0.15346071346878576)

Manhattan simulation (sims=100, 10 cities)

DescribeResult(nobs=100, minmax=(802.68295183093824, 1346.9780085984244), mean=1012.710915438476, variance=22963.525923545065, skewness=0.7302988117332708, kurtosis=-0.8227020778776559)

L infinity simulation (sims=100, 10 cities)

DescribeResult(nobs=100, minmax=(1748.0, 2643.2000000000003), mean=2072.2800000000002, variance=58205.677575757596, skewness=0.6240581762786117, kurtosis=-0.6260481610607247)