A - Simulated Annealing:

To begin the search, an initial state is randomly generated by shuffling a list of distinct consecutive integers from 1 to the size of the city file using random.shuffle from the 'random' library. The length of this initial state is calculated by evaluating the distances between each pair of cities and storing the sum - shorter tours are considered to be better solutions. Time is initialised at 0, at which point the system is at its starting temperature of 10000000 although this is subject to change during experimentation. New successor states are individually generated and their lengths are compared against that of the current state. If a successor state is accepted, it becomes the current state - shorter tours are accepted unconditionally and longer tours are accepted with some probability.

My implementation will accept worse solutions with a decreasing probability, described by an *exponential function* of the temperature, T, and the difference in tour length, ΔE, between the current and successor state ($e^{-\Delta E / T}$). A successor state that is marginally longer than the current state is more likely to be accepted in accordance with this probability function than one that is considerably longer. After each new successor state is generated, the time is incremented while the system temperature is decreased according to some cooling schedule - lower temperatures also correspond to a lesser probability of accepting a worse solution. I implemented the cooling schedule as a function of the system's current temperature, initially mimicking an exponential decay, although I will also experiment with linear and logarithmic schedules. It is crucial to the success of simulated annealing that in its final steps, inferior solutions are inadmissible; this is known as the *quenching step*. In order to implement this, the cooling schedule must maintain a low system temperature long enough for the algorithm to converge towards a minimum.

In order to generate a successor state, I swap two cities in the current state which are selected by randomly generating two distinct numbers, x and y, where 0 ≤ x, y < length of tour, t. They correspond to positions in t, such that t_x and t_y are cities at indexes x and y respectively. I assess the difference in length of the conflicting regions of the current and new tour, such that an overall negative difference indicates that the new tour is shorter and a positive difference suggests that it is longer. It is useful to check the length of the new tour first before we redundantly modify the current tour only to restore it in the event of a rejection.

- Add the distances between cities in the current tour:

$$t_{x-1}, t_x, t_{x+1}, \ldots, t_{y-1}, t_y, t_{y+1}$$

- Subtract the distances between cities in the new tour:

$$t_{x-1}, t_y, t_{x+1}, \ldots, t_{y-1}, t_x, t_{y+1}$$

Calculating the difference in tour length: - where d is a function returning the distance between two cities

$$\Delta E = [d(t_{x-1}, t_x) + d(t_x, t_{x+1}) + d(t_{y-1}, t_y) + d(t_y, t_{y+1})] - [d(t_{x-1}, t_y) + d(t_y, t_{x+1}) + d(t_{y-1}, t_x) + d(t_x, t_{y+1})]$$

Hence, length of successor state = length of current state + ΔE

Simulated Annealing - Experimentation:

Each time a parameter is changed, the algorithm is run 10000 times and the average and best tour lengths are recorded. I will first investigate the effect of the following cooling schedules on tour quality, varying the size of the constant ß.

❶ $T[t] = T[t-1] / ß$ ❷ $T[t] = \frac{T[t-1]}{1 + ß \log(1+t)}$ ❸ $T[t] = T[t-1] - ß$

1 - Exponential cooling schedule ref [1]

I hypothesise that smaller values of ß will produce better tours overall. As ß increases, worse tours are accepted with a high likelihood for a decreasing amount of time as the system spends less time in a high temperature state - this may hinder the algorithms ability to escape local minima. I experimented with ß in the range ß > 2 as the running time was exceptionally bad when 1 < ß < 2 and ß must be at least 1 so that the system gradually cools.

TOUR SIZE	012		026		058		535	
ß	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
2	70	56	1674	1505	51012	38108	121858	115270
5	71	56	1741	1529	61144	46204	134547	127319
10	73	56	1784	1559	66508	49559	139416	132458
100	81	56	1885	1620	77836	59891	147182	140037
1000	88	56	1957	1676	84776	63436	150471	142915
10000	94	56	2016	1717	89810	68389	152419	144699
100000	99	56	2057	1727	93445	70392	153484	145321
1000000	104	56	2094	1775	96483	75172	154370	145955

It is clear to see from the results that ß = 2 produces the best solutions and that small increases in ß initially correspond to a rapid decline in tour quality.

To break this down further, the table below shows the increase in average tour length when ß rises from 2 to 10 as a percentage of the total increase across all measured values of ß. For tours of size 535, over half of the growth in tour length occurs before ß reaches 10. This figure drops approximately to ⅓, ¼ and a tenth for tours of size 58, 26 and 12 respectively. These findings show that small increases in the value of ß have a greater adverse effect on the quality of the longer tours. Both average and best tour lengths continue to increase as ß rises, although this tapers off for very large values of ß. In conclusion, ß = 2 offers the best tradeoff between tour quality and running time overall.

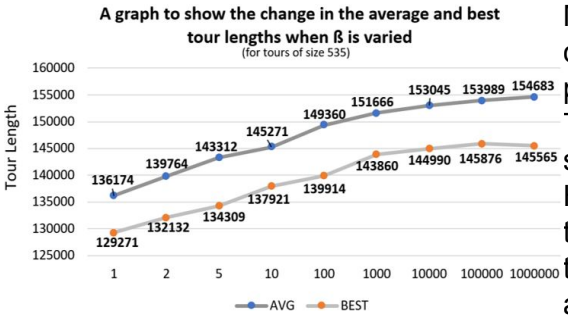
TOUR SIZE	012	026	058	535
TOUR LENGTH INCREASE (2 ≤ ß ≤ 10)	3	110	15496	17558
TOUR LENGTH INCREASE (2 ≤ ß ≤ 1000000)	34	420	45471	32512
PERCENTAGE OF TOTAL INCREASE	9%	26%	34%	54%

2 - Logarithmic cooling schedule ref [2]

I predict that small values of ß will once again produce better solutions as the temperature will decline at a slower rate than when ß is large. The value of $\log_{10}(1+t)$ will also increase as time progresses, further accelerating the speed at which the system cools. I anticipate that this will have a detrimental impact on the effectiveness of the quenching step as

temperature decline will be at is quickest in the algorithm's final steps, hence it may not settle on a local minimum.

TOUR SIZE	012		026		058		535	
ß	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
1	71	56	1753	1548	62879	44883	136174	129271
2	73	56	1787	1548	66846	51238	139764	132132
5	76	56	1828	1588	71529	52907	143312	134309
10	78	56	1853	1604	74502	56327	145271	137921
100	85	56	1932	1636	82179	61414	149360	139914
1000	92	56	1989	1694	87815	65657	151666	143860
10000	97	56	2037	1730	91803	67676	153045	144990
100000	102	56	2078	1757	94976	71144	153989	145876
1000000	106	56	2111	1740	97603	68073	154683	145565



My results show that tour quality increases in inverse proportion to the constant ß. Tour lengths are similar to schedule 1 when ß is large. However, this curve is flatter therefore small decreases in the value of ß are less effective at reducing tour length.

For this schedule, ß = 1 produces the shortest tours, though they are longer than that of the exponential cooling schedule.

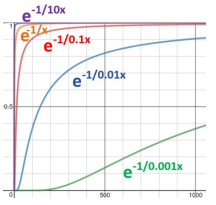
3 - Linear cooling schedule ref [1]

This schedule is unlikely to perform best overall as linear cooling schedules are unable to vary the rate of temperature decline to facilitate the improvement of solutions. For example, it is not possible to rapidly drop out of a high temperature state initially then still maintain a lengthy quenching step. My results show that decreasing ß had little effect on the tours produced, although ß = 1000 performed marginally better on average. I wanted to experiment with ß < 1000, however this was unrealistic due to the slow running time. The fluctuation in the lengths of the best tours is likely a reflection of the inherent randomness involved in simulated annealing as there is no correlation between these tour lengths and the size of the constant ß.

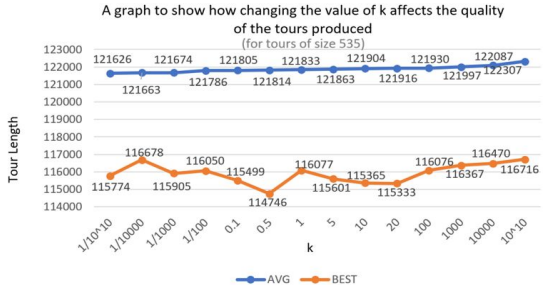
TOUR SIZE	012		026		058		535	
ß	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
1000	222	89	2677	2142	118118	90021	158835	149582
10000	223	86	2688	2107	122809	94268	159025	151378
100000	223	88	2686	2138	123559	93588	159045	148714
1000000	223	86	2685	2134	123662	95639	158997	149769

I will permanently implement the exponential cooling schedule as it produced the overall best tours. Next, I will investigate the effect of varying the constant k (where k > 0) in the probability function $e^{-\Delta E / kT}$ on tour length.

Probability of selecting an inferior tour - changing the value of k



The curve representing the exponential function $e^{-\Delta E / kT}$ flattens as k decreases, therefore the probability of keeping an inferior tour will decrease more steadily in line with temperature (though this is independent of ΔE). This is illustrated by the graph depicting multiple exponential curves. For small values of k, tours that are slightly longer are unlikely to be accepted at low temperatures but likely to be accepted for large values of k, hence some compromise may be required in order to optimise solutions.



The average and best tour lengths for tours of size 535 are illustrated in the line graph as these results best displayed the underlying pattern (changes in tour length were negligible for small tours). Increasing the value of k had little effect on average tour length although small values of k correspond to slightly better solutions. When k = 0.5, the average tour length was relatively short and the shortest 'best' tour length was produced, therefore the probability of accepting worse tours will now be fixed at $e^{-\Delta E / \frac{1}{2}T}$. I will also experiment with removing the ability to accept worse tours.

Removing the ability to select an inferior tour

Setting the probability of accepting worse tours to 0 reduces simulated annealing to a greedy hill-climbing algorithm. Accepting inferior tours provides simulated annealing with its characteristic ability to escape local minima in search of the global optimum, therefore I expect that removing this ability will correspond to worse solutions.

TOUR SIZE	012		026		058		535	
	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
probabilty = 0	78	56	1832	1604	61414	51238	127560	120501
k = 0.5	70	56	1674	1504	51065	38324	121814	114746

My results confirm this hypothesis, as both average and best tour lengths were longer than when we accept worse tours with probability $e^{-\Delta E / \frac{1}{2}T}$.

Varying the initial temperature of the system

TOUR SIZE	012		026		058		535	
Intial Temp T0	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
10^9	69	56	1674	1506	51005	38892	121843	115222
10^7	70	56	1673	1519	51032	38794	121833	115846
10^5	70	56	1674	1495	50983	37602	121824	115553
10^3	70	56	1674	1508	51034	38280	121848	115725
10	70	56	1674	1517	51073	36936	121914	116446

I propose that simulated annealing will produce better tours when initialised at a high temperature, simply because the algorithm is given more opportunities to settle on the global optimum. In order to test this hypothesis, I varied the initial temperature between 10 and 10^9, although my results show that this had almost no effect on tour length.

I decided to set the initial temperature at 10^9 as this produced relatively good average and best tours.

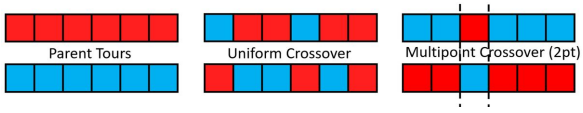
B - Genetic Algorithm:

Genetic algorithms replicate the process of evolution on a finite population of individuals - propagation of 'good' genetics will gradually rid the population of bad solutions. In the travelling salesman problem, each individual is represented by a tour and a population consists of a list of tours. A random population is generated initially by shuffling a list containing all distinct integers from 1 to the size of the tour file using random.shuffle then adding this shuffled tour to the population. This is repeated 100 times until the population contains 100 individuals who reproduce through crossover to spawn the

next generation. 5 generations are produced in total, and the fittest individual is the shortest tour to have existed at any point throughout the process of evolution.

In order to generate a child tour, two parent tours are selected for mating (fitter individuals are more likely to be chosen) using a technique known as "roulette wheel selection". The length/fitness of each individual is evaluated and stored in a list - preserving the ordering of tours exhibited in the population. The fitnesses are normalised by dividing each value by the sum of the fitnesses, such that the sum of the normalised fitnesses is 1 (this means that we are able to treat each normalised fitness as the probability of its corresponding tour's selection). To "spin" the roulette wheel, a random number between 0 and 1 is generated. We then iterate through the list of normalised fitnesses, keeping track of the cumulative total. The first tour whose normalised fitness makes the cumulative total greater than or equal to the random number is chosen.

In order to merge the genetics of two population members, I implemented two versions of crossover (*multipoint* and *uniform* crossover), each producing two children of matching length. In multipoint crossover, k 'crossover points' are selected, where $1 \leq k <$ length of tour, such that each parent is divided into k+1 segments which are alternately distributed to each child. In uniform crossover, each city in a child is independently chosen from either parent with equal probability and when a child receives a city from one parent, the other child receives a city from the opposite parent. In contrast to multipoint crossover, uniform crossover exchanges individual cities and not segments of the tour. Both implementations often produce tours in which cities are repeated and others are absent, hence the offspring must be 'converted' before we can compare their fitnesses. Duplicate cities in each child are replaced by missing cities that exist in the other child in order of appearance, forming two valid tours. The 'fittest' child is added to the new population - this is the child of shortest length. First, I will implement uniform crossover but I will also experiment with multipoint crossover.



I implemented two variations of mutation - *swap* and *scramble* mutation. In swap mutation, two distinct integers that correspond to positions in the tour are selected and the cities located there are swapped. Scramble mutation operates by reshuffling a randomly selected portion of the tour of user-defined size. Mutation creates variation in the gene pool which encourages exploration of more of the search space, however if we make the probability of mutation too high then we reduce our algorithm to a random search. Initially, I will implement swap mutation and the probability of a mutation occurring will remain constant throughout the search at 0.2, although this will vary during experimentation.

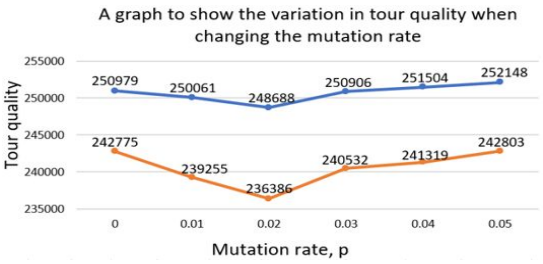
Genetic Algorithm - Experimentation:

This time, whenever a parameter is changed, the algorithm is run 1000 times and the average and best tour lengths are recorded. I will first investigate the effect of varying the mutation rate on the quality of solutions.

Vary the probability of mutation

If the mutation rate is too low, this may result in premature convergence towards a low quality solution. Conversely, if the mutation rate is too high then few good solutions will actually be preserved. Consequently, I predict that the optimal rate will be low, but that some mutation will still be necessary. Initially I varied the probability, p, between 0 and 0.5.

TOUR SIZE	012		026		058		535	
Probability, p	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
0	109	74	2134	1977	98693	90233	150043	150491
0.1	106	66	2198	1961	96998	88675	151326	148958
0.2	108	75	2153	2049	96630	91330	151437	149606



It was immediately obvious that the best results were produced between 0 and 0.1 as tour quality continued to decline after p was increased beyond 0.2. To delve into this further I decided to investigate the effects of varying p in the range $0 \leq p \leq 0.05$. The graph displays the total of the average and best tour lengths for the tour sizes 12, 26, 58 and 535 to give an overview of how the mutation rate affects tours in general. My results show a distinct improvement in tour quality when p = 0.02 which indicates that the mutation rate should be fixed at 0.02 for future experiments. This supports my hypothesis that *some* mutation is required, however too much mutation is ineffective.

Comparison of mutation techniques - scramble & swap mutation

I will compare the quality of solutions when implementing swap and scramble mutation. To make it possible to compare tours of different sizes, I will scramble a *percentage* of each tour. I predict that mutation that preserves most of the tour will perform better (e.g. swap mutation or scramble mutation of a low percentage). Overall, changing the type of mutation did not affect the results all that much, although the general trend is such that tour quality improved as the percentage of the tour that was scrambled decreased. There are some exceptions to this as when 30% of the tour was scrambled, the worst average tour lengths were produced for tours of size 12, 26 and 58. However, average tour length was good for tours of size 535 and the shortest 'best' tour was found. Swap mutation performed best for mid-sized tours and relatively well for small and large tours, therefore I decided to implement this permanently.

TOUR SIZE	012		026		058		535	
	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
Swap	107	73	2056	1969	98306	88651	151530	149399
5% Scramble	105	82	2080	2035	98836	88824	151372	148924
10% Scramble	106	81	2075	2018	98503	88488	151136	148429
15% Scramble	107	83	2078	2025	99284	90926	151451	147728
20% Scramble	105	77	2110	2051	99799	92129	151422	148479
25% Scramble	107	87	2106	2057	101626	92530	151449	149169
30% Scramble	107	83	2203	2059	102406	93000	151347	148245

Exclusively accepting better mutations

ref [3]

In this experiment, the mutation rate will remain 0.02, however a tour is only mutated if this results in an improved tour.

The following table contains the results of this experiment, as well as the results from the previous experiment for comparison. They show that accepting all mutations is more effective at improving solutions for tours of size 58, however shorter tours were produced when exclusively accepting better mutations for tour sizes 12, 26 and 535. I decided to keep this adjustment as it resulted in better solutions overall for the majority of tours.

TOUR SIZE	012		026		058		535	
	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
Control	107	73	2056	1969	98306	88651	151530	149399
Experiment	105	70	2183	1968	98597	87138	151182	148262

Comparison of crossover techniques - uniform crossover & multipoint crossover

In uniform crossover, there is no guarantee that two neighbouring cities will be inherited together, even though they are likely to be a small distance apart in short tours. Hence I hypothesise that multipoint crossover will outperform uniform crossover as I predict that lacking this characteristic will be detrimental to the search. In order to test this theory, I will vary the number of crossover points, p, in multipoint crossover and compare the results against that of uniform crossover.

TOUR SIZE	012		026		058		535	
Crossover Type	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
Uniform	107	73	2056	1969	98306	88651	151530	149399
Multipoint, p = 1	105	61	2173	1955	97753	86420	150977	148349
Multipoint, p = 2	106	62	2174	1890	97705	84940	150881	146733
Multipoint, p = 3	107	62	2175	1956	97520	84801	150945	148427
Multipoint, p = 4	109	67	2173	1958	97562	84478	151008	146036
Multipoint, p = 5	109	69	2176	1969	98338	84954	151057	148512
Average tour length	107	65.67	2155	1950	97864	85707	151066	147909
Multipoint, p = 2	106	62	2174	1890	97705	84940	150881	146733
% better	1%	6%	-1%	3%	0%	1%	0%	1%

My results support my hypothesis as multipoint crossover performed better on most occasions, though it continued to become less effective as p grew beyond 5. 2-point crossover appears to give good average and best tours for a *range* of tour sizes, therefore I will implement this technique permanently. I calculated the percentage by which the tours produced using 2-point crossover were better than that of the other techniques as a whole. I found that 2-point crossover's best tours were 3% better than the average of the best tours produced by the other techniques.

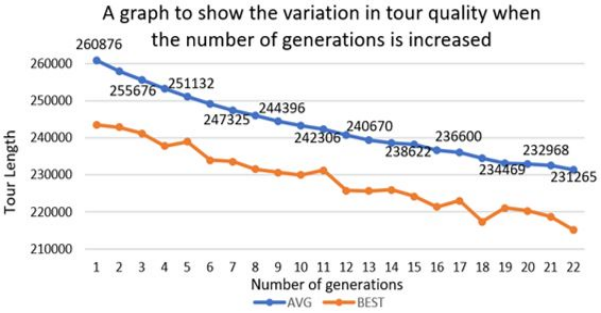
% better overall	
BEST	3%
AVG	0%

Implementing a "crossover rate" so that not all members are crossed over

TOUR SIZE	012		026		058		535	
Crossover Rate, p	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
p = 1	106	62	2174	1890	97705	84940	150881	146733
p = 0.8	109	73	2184	1967	98173	82053	151035	148405
p = 0.6	112	79	2194	1939	98857	85139	151244	148738
p = 0.4	114	70	2210	2034	99776	88224	151630	149167
p = 0.2	119	67	2236	1987	101161	83507	151914	148891

The crossover rate can be likened to the "aggressiveness" of the search as a high crossover rate generates more new offspring whereas a low crossover rate preserves good chromosomes through the generations. Resource [4] indicates that in order to produce good solutions, approximately 70% of parents should undergo crossover. However, my results indicate that a 100% crossover rate is most effective for the city files provided.

Vary the number of generations



I hypothesise that increasing the number of generations will result in better solutions as individuals in each new generation are fitter on average than individuals in the previous generation. In order to test this hypothesis I varied the number of generations between 1 and 22. My results show that producing more generations improved the quality of solutions, although the running time became very slow. As a result of this, I was unable to increase the number of generations beyond 22, even though I expect that my results would continue to improve.

Best tours obtained for each tour size by SA (simulated annealing) and GA (genetic algorithm):

Tour Size	012	017	021	026	042	048	058	175	180	535
Length of best tour - SA	56	1444	2549	1475	1131	14382	25395	23974	2820	55311
Length of best tour - GA	56	1444	2739	1523	1229	16571	32116	42312	270760	143594

In conclusion, simulated annealing had a far quicker running time and also produced better tours. In order to get the final results for simulated annealing, the best tour so far was made the initial tour at the beginning of each iteration and the temperature was lowered in order to reduce the amount of deviation from the current shortest tour. I modified my genetic algorithm by increasing the number of generations to 10000 and I allowed it to run overnight. As a result there are vast improvements between these tours and those produced by experimentation.

References:

[1] www.fys.ku.dk/~andresen/BAhome/ownpapers/permanents/annealSched.pdf (page 3)
[2] what-when-how.com/artificial-intelligence/a-comparison-of-cooling-schedules-for-simulated-annealing-artificial-intelligence/
[3] https://www.researchgate.net/post/Why_is_the_mutation_rate_in_genetic_algorithms_very_small
[4] http://www.ai-junkie.com/ga/intro/gat2.html