Report

Name: Carl Torgny Helin
Username: carlthe

Exhaustive Search

The shortest route I found among the first 10 cities are

['Barcelona', 'Dublin', 'Brussels', 'Hamburg', 'Copenhagen', 'Berlin', 'Budapest', 'Belgrade', 'Bucharest', 'Istanbul'] or ['Barcelona', 'Istanbul', 'Bucharest', 'Belgrade', 'Budapest', 'Berlin', 'Copenhagen', 'Hamburg', 'Brussels', 'Dublin'] or any other combination with the cities in the same order

The program took about 7.15 seconds to find out the answer for 10 cities

n	time	increase multiplied
5 cities	= 0.00026702880859375 seconds	0.91503268
6 cities	= 0.001422882080078125 seconds	5.328571429
7 cities	= 0.006685972213745117 seconds	4.698894102
8 cities	= 0.05447983741760254 seconds	8.148379275
9 cities	= 0.5113012790679932 seconds	9.385146933
10 cities	= 7.15315318107605 seconds	13.99009444
11 cities	= 88.60693502426147 seconds	12.38711555

It’s clear that the time it takes increases exponentially, the reason for this is because the possibilities for one more city increases the possible routes factorial.

	factorial
Possible routes for 6 cities are 720	6
Possible routes for 7 cities are 5,040	7
Possible routes for 8 cities are 40,320	8
Possible routes for 9 cities are 362,880	9
Possible routes for 10 cities are 3,628,800	10
Possible routes for 11 cities are 39,916,800	11
Possible routes for 12 cities are 479,001,600	12
Possible routes for 13 cities are 6,227,020,800	13
Possible routes for 14 cities are 87,178,291,200	14
Possible routes for 15 cities are 1,307,674,368,000	15
...	
Possible routes for 24 cities are 620,448,401,733,239,439,360,000	24

Based on that 11 cities took over a minute and 12 cities took too long we can definitely assume that if we are testing this method for 24 cities its going to take a while. This problem can actually translate to a brute force problem, since we're also almost trying all possible permutations as well.

According to this website https://tmedweb.tulane.edu/content_open/bfcalc.php if we try 475,920,314,814,253,359,955,968 combinations (of passwords) instead of the 620,448,401,733,239,439,360,000 combinations as in our example it will take 384,752,893,146.37 days or 1 million year-ish, but this is based on a typical PC processor in 2007 and that the processor is under 10% load.

Hill Climb

The hill-climber created starts in random city, then will always travel to its closest neighbour until it's no more neighbours left, which it needs to travel all the way back to its origin. This might become a problem in some cases since it's not considering smart choices on how to get all the way back to start again. I've figured out that the hill limber performance is well, not optimal as the exhaustive search, but sometimes manages to find the shortest route. Underneath you can see my results running the algorithm on all the cities and ten cities.

10 cities, run 20 times
Best 7680.49 km
Worst 8309.61 km
Mean 7939.1885 km
Standard deviation 218.4215817 km

24 cities, run 20 times
Best 12095.08 km
Worst 14218.74 km
Mean 13270.619 km
Standard deviation 638.6070744 km

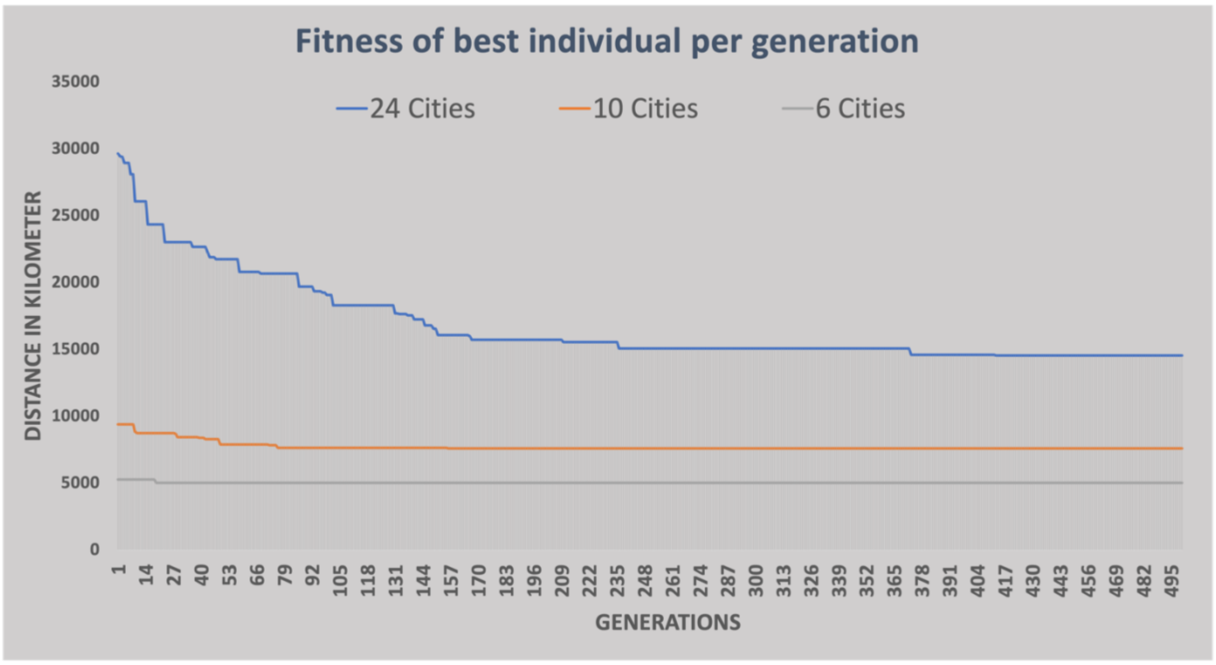
Genetic Algorithm

I've chosen to make my genetic algorithm with focus on elitism over exploration which may not be as useful in other scenarios where the global optima could be more spread out. In this case however I'm satisfied with the result I'm getting by exploring deeper into a good solution I've encountered. I'm exploring 6 cities with only ten routes in the population, 10 cities with 100 routes in the population and a thousand routes for 24 cities to first initialize a starting point. I use the PMX algorithm as a mutation operator where parent A gives a chunk of genetic material and parent B spreads its across over parent A's to create a child. My crossover operator is simple and based on randomness, where two positions in the chromosome change values. I experienced a bit with more exploration to replace certain good solutions with worse ones based on randomness but figured out that this mostly led to a worse end-result, but this is still an option in the program if you change one line of code.

Cities	24	10	6
Population	1000	100	10
Generations	700	100	10
Best	12722.56 km	7486.31 km	5018.81 km
Average	13911.18 km	7792.08 km	5374.33 km
Worst	15703.06 km	8444.45 km	5778.38 km
Standard Deviation	798.96 km	322.37 km	290.36 km
Time	0.06 seconds	0.0075 seconds	0.0005 seconds

This is my result from running the algorithm on three different populations 20 times. The speed is pretty good compared to exhaustive search, but slower than hill-climbing. The shortest routes it found is correct for both ten and six cities. The number of tries the EA algorithm depends on how many generations we decide to run it with and how many routes to put into the original population. At this moment without changing variables the algorithm will inspect 600 routes among 10 cities if its starting population is 100 routes and running with 100 generations. In comparison to exhaustive search inspecting 3,628,800, my algorithm inspects 6048 times fewer routes.

Underneath you can see I've plotted 500 generations run on 6, 10 and 24 cities with corresponding population sizes as 10, 100 and 1000. I've figured out that the algorithm needs more generations to find the optima when trying to figure out the shortest route for 24 cities.


Fitness of best individual per generation
24 Cities
10 Cities
6 Cities
DISTANCE IN KILOMETER
GENERATIONS