ECMM409 – Individual Coursework   Alexander Wharrier

Introduction

Welcome to my code report for the Individual coursework assignment for ECMM409 Nature-Inspired Computation. The goals of this assignment are to implement an EA to optimise the Travelling Salesman Problem for cost. For this coursework I decided to use Rust as my programming language for its speed and concurrency capabilities. I will include binaries compiled for Windows and Linux in my Zip file and if you wish to compile it yourself (or run it on Mac) there are instructions in the README.md file to do this, though you will need rust installed on your system to do so.

The requirements set out for this program are: The code finishes execution after 10,000 fitness evaluations. As there is a fitness evaluation every generation this in essence means 10,000 generations; The user can specify a population size that is randomly generated; The code uses Tournament Selection with a user selected tournament size; The code implements crossover with fix and ordered crossover; The code implements inversion, single swap and multiple swap mutation and the code implements a replacement function discussed in our lectures – in this case I chose to use the Replace Weakest function.

Due to Rust being a relatively recent programming language, there are very few crates (libraries in rusts ecosystem) dedicated to EA. Therefore I wrote my own for this program. All my code that is actually compiled is present in the src directory. I have 6 library files: country.rs, chromosome.rs, population.rs, simulation.rs, interface.rs and lib.rs. Then I have main.rs to actually use and run this code. Integration tests for my code are located in the test directory and are not compiled into the final code so can be ignored.

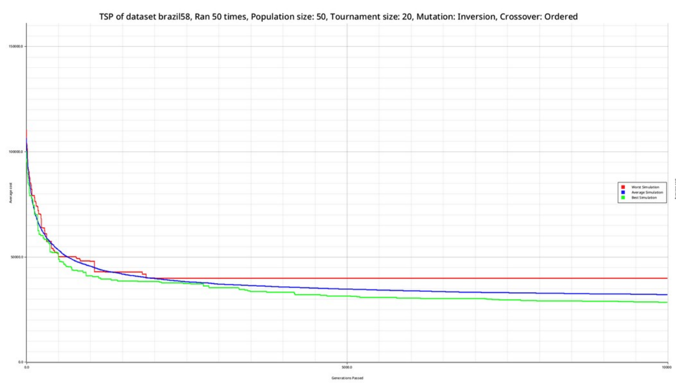
As Rust is a compiled language with notorious compile times, I added a command line interface so that I can use one binary to run all my experiments without having to recompile between each run. The flags to control what options are selected are documented in the README.md, the interface.rs file and in the help flags of my programs CLI. As I knew that the initial populations randomness had an effect and due to not being able to specify the rng seed (its a lazily-initialized thread-local random number generator, seeded by the system) I decided to use multi-threading with a user selected number of runs for each dataset to run concurrently so that multiple runs of each type can be compared together and an average can be taken. Due to Rust's speed, running 50 simulations each for burma14 and brazil58 with a population size of 50 and tournament size of 20, ordered crossover and inversion mutation takes roughly 46 seconds on my AMD Ryzen 5 5500U laptop cpu. For all my experiments I therefore elected to run 50 simulations for each dataset for each experiment so that I can show not only the best convergence, but also the average convergence of the group to hopefully reduce the impact of initial rng on the comparison of the parameters.

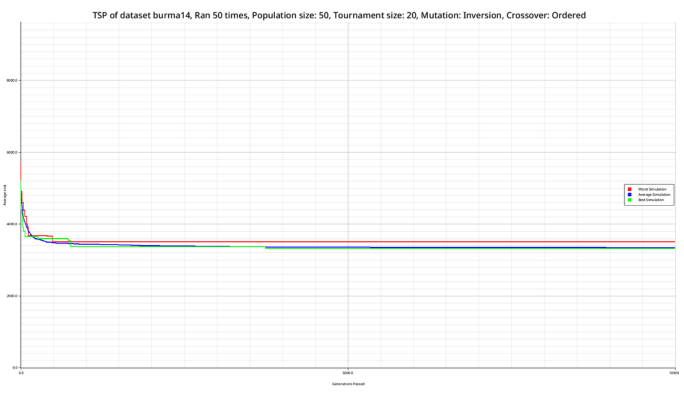
Experimental Results

Comparing Mutation

Inversion

Command: ./tsp-coursework -p 50 -t 20 -n 50 -m I -c O -o R -s B
Final Results – Ran in 45 seconds
burma14 cheapest cost = 3,323, burma14 average end cost = 3,343.28
brazil58 cheapest cost = 28,560, brazil58 average end cost = 32,129.74

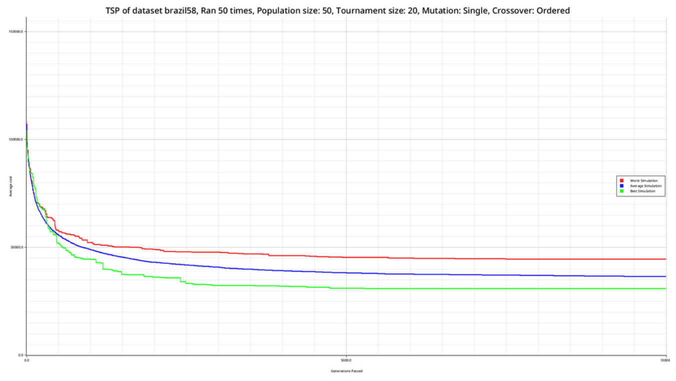


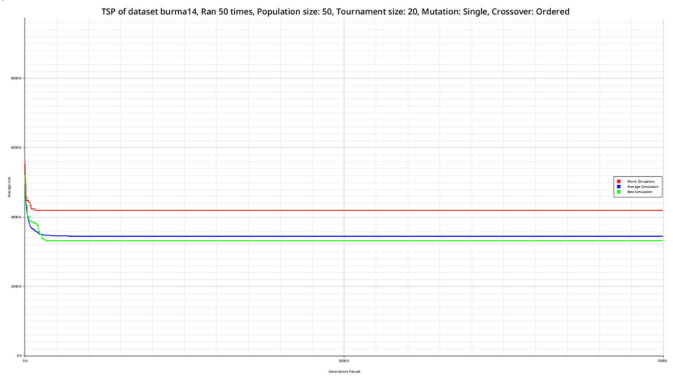


Single Swap

Command: ./tsp-coursework -p 50 -t 20 -n 50 -m S -c O -o R -s B
Final Results – Ran in 46 seconds
burma14 cheapest cost = 3,323, burma14 average end cost = 3,444.94

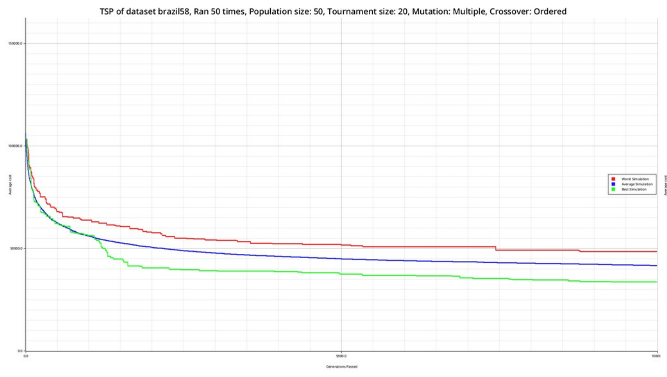
brazil58 cheapest cost = 30,859, brazil58 average end cost = 36,551.86

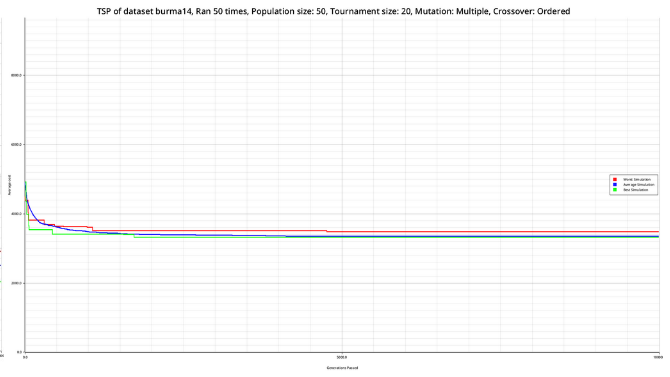




Multiple Swap

Command: ./tsp-coursework -p 50 -t 20 -n 50 -m M -c O -o R -s B
Final Results – Ran in 46 seconds
burma14 cheapest cost = 3,323, burma14 average end cost = 3,350.80
brazil58 cheapest cost = 33,730, brazil58 average end cost = 41,740.16

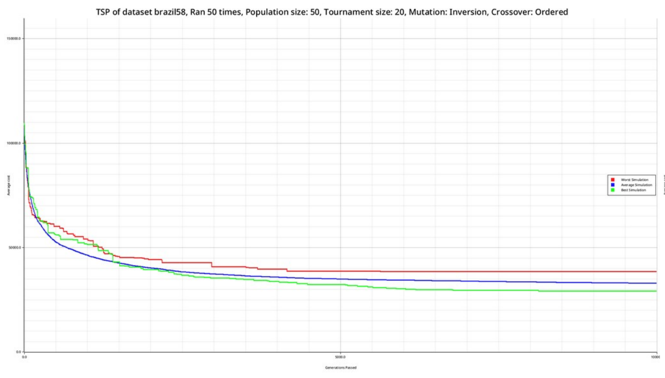


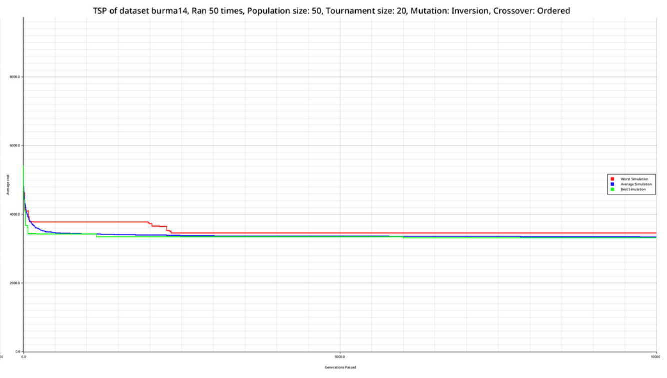


Comparing Population Size

Population of 50

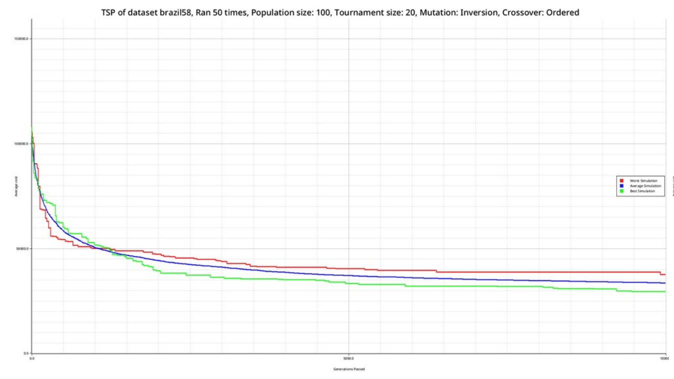
Command: ./tsp-coursework -p 50 -t 20 -n 50 -m I -c O -o R -s B
Final Results – Ran in 46 seconds
burma14 cheapest cost = 3,323, burma14 average end cost = 3,340.38
brazil58 cheapest cost = 29,229, brazil58 average end cost = 32,964.09

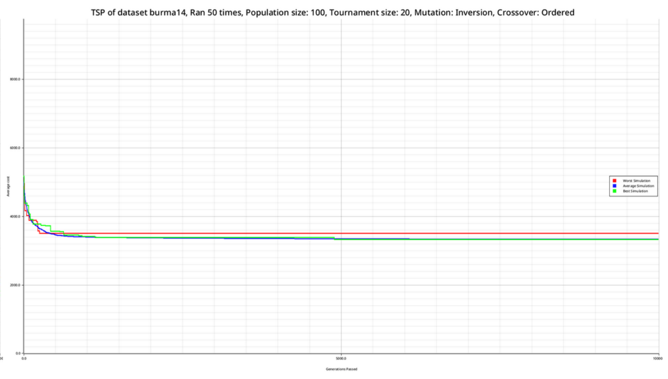




Population of 100

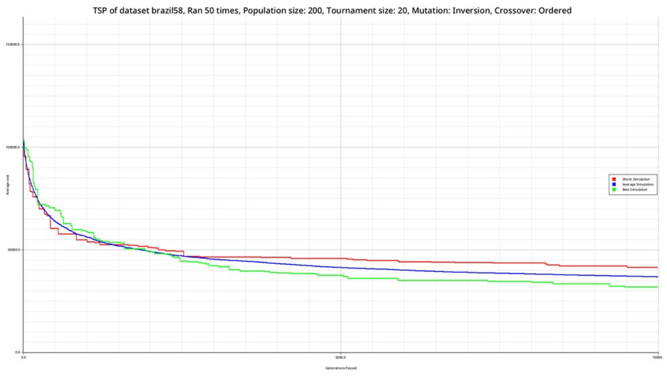
Command: ./tsp-coursework -p 100 -t 20 -n 50 -m I -c O -o R -s B
Final Results – Ran in 46 seconds
burma14 cheapest cost = 3,323, burma14 average end cost = 3,333.88
brazil58 cheapest cost = 29,484, brazil58 average end cost = 33,644.02

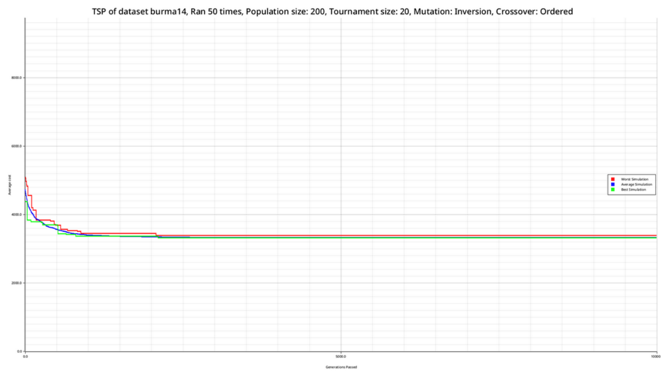




Population of 200

Command: ./tsp-coursework -p 200 -t 20 -n 50 -m I -c O -o R -s B
Final Results – Ran in 49 seconds
burma14 cheapest cost = 3,323, burma14 average end cost = 3,329.54
brazil58 cheapest cost = 31,979, brazil58 average end cost = 36,836.55

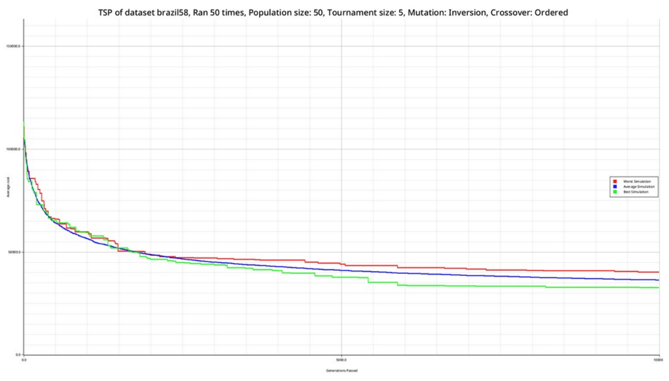


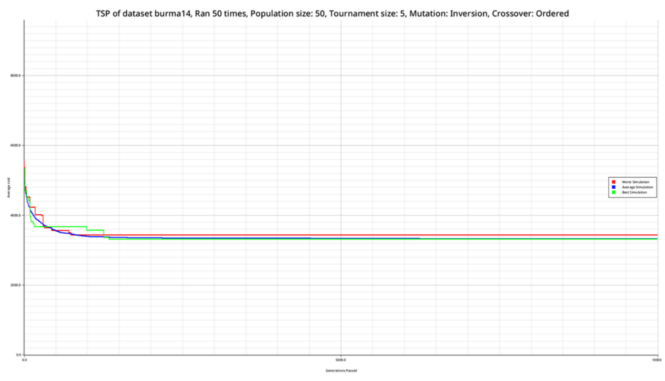


Comparing Tournament Size

Tournament Size of 5

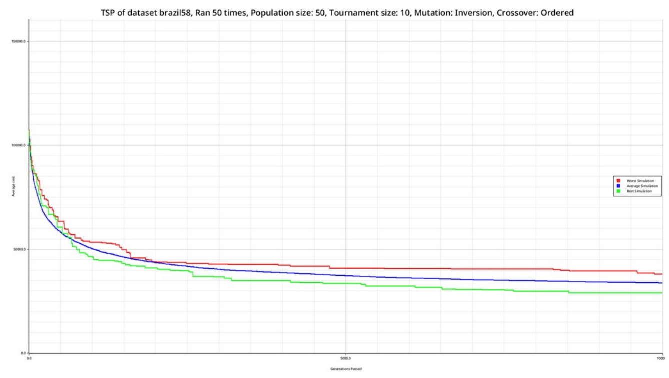
Command: ./tsp-coursework -p 50 -t 5 -n 50 -m I -c O -o R -s B
Final Results – Ran in 48 seconds
burma14 cheapest cost = 3,323, burma14 average end cost = 3,330.28
brazil58 cheapest cost = 32,777, brazil58 average end cost = 36,477.16

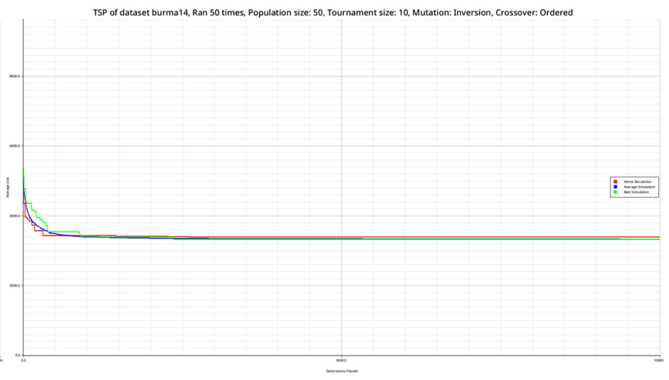




Tournament Size of 10

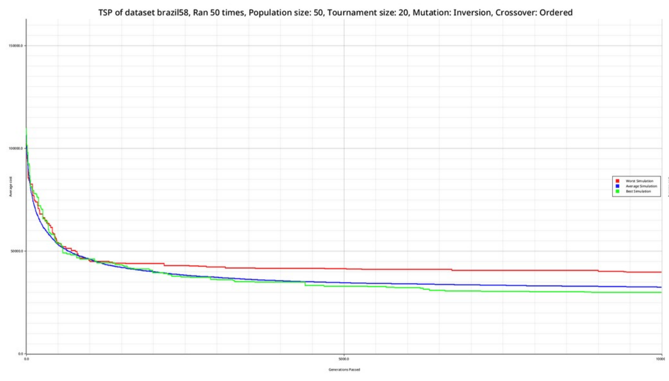
Command: ./tsp-coursework -p 50 -t 10 -n 50 -m I -c O -o R -s B
Final Results – Ran in 46 seconds
burma14 cheapest cost = 3,323, burma14 average end cost = 3,327.28
brazil58 cheapest cost = 29,067, brazil58 average end cost = 33,883.32

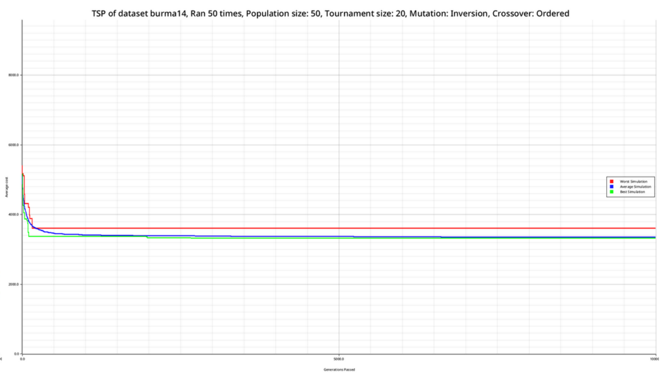




Tournament Size of 20

Command: ./tsp-coursework -p 50 -t 20 -n 50 -m I -c O -o R -s B
Final Results – Ran in 47 seconds
burma14 cheapest cost = 3,323, burma14 average end cost = 3,351.28
brazil58 cheapest cost = 29,971, brazil58 average end cost = 32,548.45





Conclusions

Both crossover methods showed extremely similar results so they're graphs have been omitted for space. Ordered crossover has a slightly cheaper best simulation for brazil58 of 29,061 vs 29,695 and a slightly cheaper overall average simulation of burma14 of 3338.12 vs 3,373.54 but a slightly more expensive overall average simulation of 32,399.20 vs 32,249.74. With this overall slight advantage I believe Ordered Crossover to be marginally better

Inversion mutation is shown to be the best the best mutation type as it produced the best overall simulation with a score of 28,560 for brazil58, and its average output after 10,000 generations for brazil58 was also the lowest at 32129.74. Population size of 50 is shown to be a clear winner as it produced the best overall simulation with a score of 29,229 for brazil58 and its average output after 10,000 generations for brazil58 was also the lowest at 32,964.09 Tournament size was is shown to be marginally better when at 10, having a lower brazil58 cost of 29,067 and lower burma14 average output of 3,327.28. However the average simulation output for brazi58 was slightly higher than tournament size 20 at 33,883.32 vs 32,548.45.

Looking at all these results, the most optimal parameters to run this program appear to be Inversion Mutation, Ordered Crossover, Population Size 50 and Tournament Size 10. However initial population seems to effect the results, even after running 50 simulations and 10,000 generations. The run under "Inversion Mutation" with a tournament size of 20 outperformed the run under "Tournament size of 10" even though the commands were identical apart from the tournament size, so perhaps the difference is too close to tell between 10 and 20. Still, I believe this combination is most effective for a number of reasons. It has a small population and I am using replace weakest function – so only 2 chromosomes are replaced per generation. Also the lower tournament size reduces the chance of falling into a local minima. Inversion mutation has the chance to introduce much more mutation to a chromosome than single or multiple swap mutation and ordered crossover keeps the order of parts of the parents the same, which is most effective in TSP type problems where order is important.

A good heuristic function to add would be local search, as it would significantly speed up solving the problem. A variation of this algorithm that might be better would be to use an elitist replacement and replace a larger amount of the population each time. This way larger population sizes could be used and they would converge faster – with steady state even using a population of 200 the cost curve takes noticeably longer to decrease than the population of 50. Finally, another nature inspired algorithm that may provide better results is Ant Colony Optimisation. This is because TSP naturally aligns with ant path finding behaviour.

References

1. Brownlee, J. (2012). *Clever algorithms : nature-inspired programming recipes*. United Kingdom: Lulu.com.

2. Eiben, A.E., Smith, J.E. and Springerlink. (2015). Introduction to Evolutionary Computing. Berlin, Heidelberg: Springer Berlin Heidelberg.

3. Davis, L. (1991). Handbook of Genetic Algorithms. Van Nostrand Reinhold Company.