Optimisation of Travelling Salesman Problem using Simulated Annealing and Genetic Algorithms

I. INTRODUCTION

The Travelling Salesman Problem (TSP) asks given n cities and their relative distances from one another what is the shortest route or tour for a salesman to the all cities starting and ending at the same point. 10 TSPs, ranging from 12-535 cities, were optimised with no explicit emphasis on the running time.

Although simple to conceptualise, the number of possible tours grows as $(n-1)!/2$ [1] and so to solve exactly via global search strategies, such as A* search, takes an impractically long period of time. Therefore local search strategies, namely simulated annealing (SA) and genetic algorithms (GA), were employed.

States were defined as path representations of tours, the objective function f defined as the length of the tour and state-to state transitions defined by the search strategy used. City distances were represented by a nxn symmetric matrix with the coordinates of a cell giving the cities and the cell value the distance. Objects representing TSPs and states were implemented.

This report outlines each algorithm's implementation details and the experimental process undertaken to optimise them. Finally the results from each algorithm are be presented.

II. SIMULATED ANNEALING

Initially the SA algorithm was based on the original work by Kirkpatrick et al [2]. An initial random tour, x, was generated and an initial temperature, t_0, given. From x a neighbour y was formed by a specified neighbourhood structure, in this case two cities in x's tour were swapped. Then the Metropolis algorithm, originally used to model the random movement of atoms up as well as down a potential well, was applied. The lengths of x and y were evaluated and if y was shorter it replaced x. However, even if y was longer there was still a probability, p, that x could be replaced. This was given by $exp(-(f(y)-f(x))/t)$, where t is defined as the current temperature. By Monte Carlo sampling, y replaced x if a randomly generated number, r, in [0,1] was less than p. The Metropolis algorithm was then run m times known as the Markov chain length. Then t was decreased according to a predefined schedule. In this case a geometric series given by $t_{i+1} = \alpha t_i$, where $\alpha = 0.999$. The whole algorithm was looped c times. The current best tour was held as a variable and returned when all loops terminated.

From this implementation it was seen that to optimise the SA algorithm investigations into the neighbourhood selection, temperature schedule and loop length along with the parameters t_0 and α were required.

Firstly a greedy hybrid operator was found in the literature as an alternative neighbourhood structure [3]. It defines three operators; an inverse which inverts a random subtour, the swap operator from the original implementation and an insert operator which moves a random subtour to a new random position. All three of these operators are run and the resulting smallest tour is returned as the neighbour. This operator was implemented and the resulting increase in performance can be seen Fig.1A.

Secondly, the number of loops m and c where considered. For the final runs the running time was not a factor so a condition of 250 consecutive best tours or $t < 2$ was given for the outer loop to terminate. For testing purposes $c = 1000$ to reduce running time. For the inner loop the literature suggests a connection between the number of cities and m, such as $m = 2n$ [4]. However it was found for small n this led to convergence to local minima as not enough loops where performed at high temperatures. As running time was not being optimised m was fixed at 800, unless $2n$, was greater to prevent this issue.

Next t_0 and α were investigated. To find the optimum values, assuming independence, multiple runs for all TSPs were required over a range of parameter values to produce a large enough data set from which to draw statistically significant conclusions. However, the number of runs required to account for the random nature of the search, differing topology of the TSPs and sensitivity of the SA to parameter change was impractical due to constraints on time and computing power.

This parameter optimisation issue has also been identified in the literature and lead to development of methods such as adaptive cooling schedules. In a recent journal article Zhan at el [4] outlined a method called List-Based Simulated Annealing (LBSA). They claimed their relatively simple adaptation of SA gave competitive robust performance over a wide range of initial parameter values through an adaptive cooling schedule. LBSA was implemented.

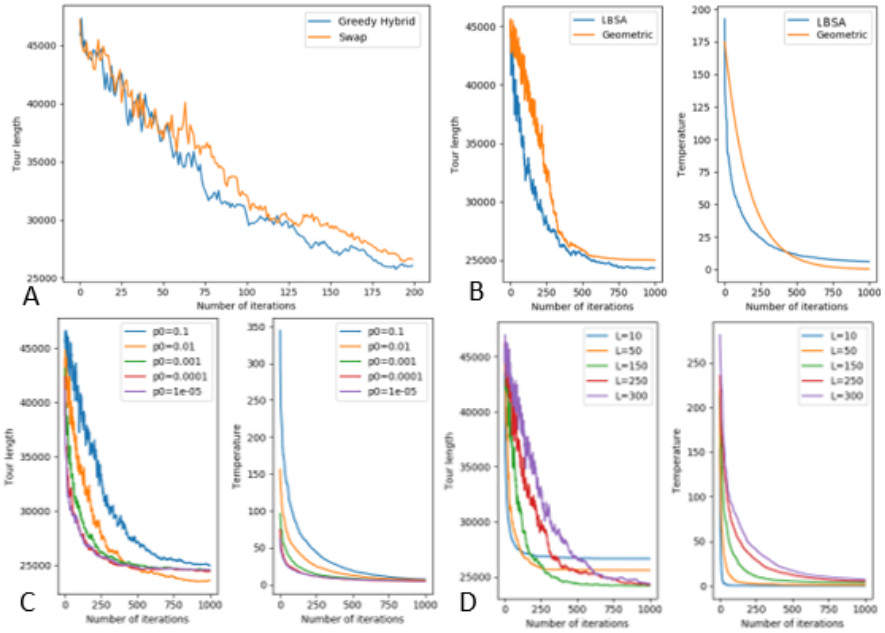

FIG. 1: Graph A shows the comparative performance of the greedy hybrid operator and swap operator. Graph B shows the increased performance of LDSA compared to SA with a geometric cooling schedule. Graphs C and D show the effect of p_0 and L on the LBSA algorithm respectively. All tests done on 175 city TSP due to its balance of computation time and complexity and run for $c = 1000$

Instead of stipulating t_0 an initial list, T_L, of temperatures, of a user defined length L, was produced along with a random state, x and neighbour, y, y replaced x if it was shorter. A temperature, t_i, was then given by $t_i = -|f(y) - f(x)|/ln(p_0)$ where p_0 is a user defined probability. t_i was added to T_L. This was then looped L times. T_L was a priority queue serving the highest temperature first as t_{max}. SA was then run as normal. For each Markov chain if a worse state was accepted a new temperature was produced and stored in a list, T_m, using the same formula as for t_i however p_0 was replaced by r. At the end of a Markov chain if the list of new temperatures was not empty t_{max} was popped from T_L and replaced by the average of T_i. Although more complex LDSA gives adaptive temperatures tailored to the topology of each TSP. Fig.1B shows the improved performance of LDSA compared to SA.

Finally the independence of p_0 and L was tested. Fig.1C shows for p_0 although the starting temperatures increase with decreasing p_0 the adaptive nature of the temperature schedule means the resultant tour lengths were invariant. Fig.1D shows that when L is too small it leads to rapid convergence to a local minima. Hence a large value for L is required. From this data p_0 and L where chosen to be 0.1 and 200 respectively, in line with Zhan at el [4].

In conclusion, LDSA was implemented with a greedy hybrid operator as a neighbourhood structure, $m = 800$ or $2n$, $p_0 = 0.1$, $L = 200$ and was run until the best tour was constant for 200 outer loops or $t < 2$.

III. GENETIC ALGORITHM

The GA implementation began by producing a random initial population of a user defined size. From this population two parents were selected with a bias towards those with shorter tour lengths. This bias was created using a cumulative probability list. Probabilities were calculated by subtracting the index's tour length from the largest tour length and then dividing by the sum of the population's tour lengths. To then choose a parent a random number, r, was generated and the cumulative probabilities were then iterated over from low to high and if the value at an index was less than r that index was selected from the population list. Using the two selected parents a child was then produced by a crossover operator. Then with a fixed probability, m, the child could have been mutated by another operator. If the worst tour in the population had a tour longer than that of the resulting child it was replaced. This was then looped c times after which the best in the population was returned. The production of only one child per iteration follows the principles of the GENITOR algorithm [1]. The opportunities to optimise this algorithm came from investigating different mutation and crossover operators and the mutation and population parameters.

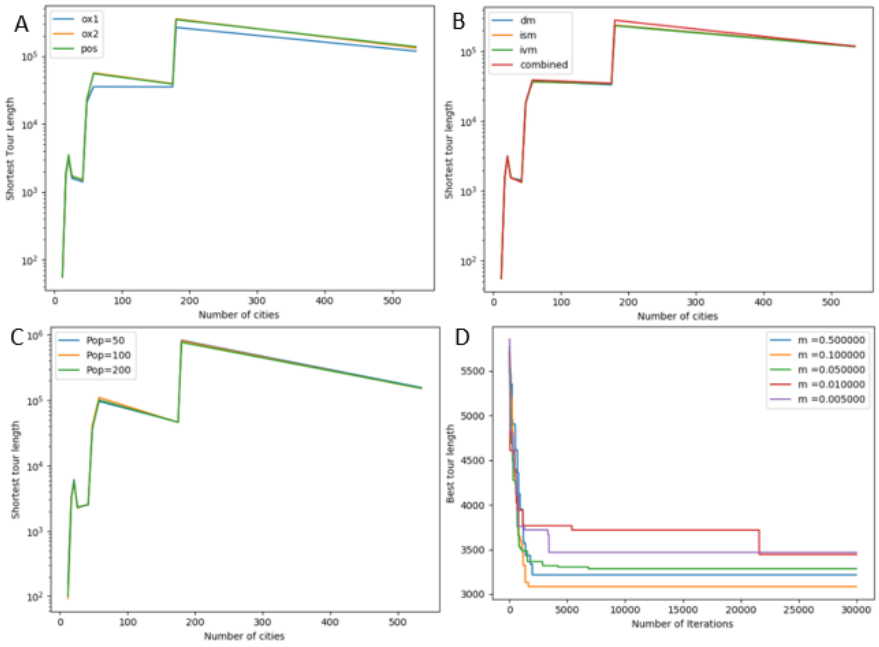

FIG. 2: Graph A, B and C show the effect of differing crossover operators, mutator operators and population sizes respectively. The data was produced using 10000 iterations. Graph D shows the effect on convergence of varying m, on the 21 city TSP

Firstly, 3 different crossover operators were investigated. In their review of genetic algorithms Larranaga et al provided empirical evidence showing order crossover, OX1, order based crossover, OX2, and position based crossover, POS, operators gave the best tours if time constraints were ignored. However, their relative performance depended on the topology of the TSP [1]. All three were implemented and tested. In OX1, a random subtour is spliced from the father. The remaining indices are then filled from the mother starting from the RHS splice index, skipping already added cities. In OX2 a number of random indexes are chosen in the father and the cities in these positions are deleted in mother tour and replaced in the order of their appearance in the father. Finally in POS a number of random cities are selected from the father and the rest of the indices are filled from the mother's tour starting at index 0 and skipping already added cities. Fig.2A shows the results from running the 3 operators on the set of TSPs for 1000 iterations. It can be seen OX1 gave the best tour lengths and so was used in the implementation.

Secondly, mutator operators were investigated. Again the work by Larranaga et al [1] showed empirically 3 operators outperformed all others. The displacement mutation, DM, and inversion mutation, IVM, both remove and reinsert a random

subtour at a random index. IVM also inverts the subtour. The final operator is the insertion mutation, ISM, in which a single city is randomly removed and reinserted [1]. Mimicking the neighbourhood structure used in SA another operator was implemented in which all three previous operators were run and the smallest mutated tour was kept. These 4 operators were tested on all 10 TSPs for 10000 iterations, the results of which can be seen in Fig.2B. There was very little difference between the operators but IVM was marginally superior especially for the 180 city instance and so was used in the final implementation.

Finally the values for population size and m were optimised. The GA was run for 10000 iterations for the set of TSPs, Fig.2C shows that a population of 200 gave the best tours. When testing mutation rate as mutations avoid convergence to local minima a converging data set was needed, so a low n TSP was used. Fig.2D shows that the 0.1 rate was optimal. Like the SA algorithm, the algorithm was set to terminate after 10000 iterations with a constant best tour.

In conclusion, a GENITOR inspired GA was implemented using OX1 crossover operator, IVM mutator operator, a population size of 200 and a mutation rate of 0.1 and run until the best tour was consistent for 10000 iterations.

IV. RESULTS

The SA and GA algorithms were run for each TSP 4 times and the best results are recorded in Table I. It is of note that the GA performed well at low n and the SA at high n. A reason for this may be for small n, the LDSA algorithm used gave very low initial temperatures for low n which quickly falls off leading to convergence to local minima.

Number of Cities	Simulated Annealing	Genetic Algorithm
12	**56**	**56**
17	1531	**1456**
21	2847	**2728**
26	1513	**1502**
42	1152	**1103**
48	**14309**	15726
58	**27607**	29266
175	**23547**	25641
180	**6600**	39520
535	**63566**	67733

TABLE I: Output tour lengths given by SA and GA for 10 given TSPs. Bold numbers are the shortest tour out of the two algorithms

References

[1] P. Larranaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review, 1999.

[2] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. Science, 1983.

[3] ChagYing Wang, Min Lin, YiWenZhong, and Hui Zhang. Solving travelling salesman problem using multiagent simulated annealing algorithm with instance-based sampling. International Journal of Computing Science and Mathematics, 2015.

[4] Shi hua Zhan, Juan Lin, Ze jun Zhang, and Yi wen Zhong. List-based simulated annealing algorithm for traveling salesman problem. Computational Intelligence and Neuroscience, 2016.