Using Particle Swarm Optimisation on the Travelling Salesman Problem

Bhuvan Setty
bhuvi.setty@gmail.com
University of Exeter
Exeter, United Kingdom

ABSTRACT

In this paper, the PSO algorithm will be implemented on various TSP instances. The PSO will be adapted, so the algorithm will be able to solve the TSP problems. The parameters of the PSO will be finely tuned, and the process of the parametric decisions will be displayed clearly. Furthermore, the PSO will be compared to two other algorithms which will contextualise its performance alongside other evolutionary algorithms.

CCS CONCEPTS

• **Random Search**; • **Stochastic Hillcimber**; • **Particle Swarm Optimisation**; • **Travelling Salesman Problem**;

KEYWORDS

Travelling Salesman Problem, Particle Swarm Optimisation, Parameter Tuning, Stochastic Hillclimber, Random Search, Evolutionary Algorithms, Fitness, Optimal Path

ACM Reference Format:
Bhuvan Setty. 2018. Using Particle Swarm Optimisation on the Travelling Salesman Problem. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In this article, there will be algorithms used to attempt to solve the travelling salesman problem (TSP). Out of the various algorithms that will be used, the main focus of the paper will be the particle swarm optimisation (PSO) algorithm. The article will explain the algorithm in detail and why this algorithm was chosen to tackle the TSP problem. Then the article will explain the TSP as a problem in more detail and the difficulties of attempting to solve the problem. Subsequently the article will explain how the PSO will be adapted so it can be used on the TSP by explaining some of the problem specific representations and specific search operators our PSO will use on the TSP problem instances. After this, the article will explain the experimentation that took place to test the PSO, by explaining some of the specific instances of the problem that were explored, the parameters that were tuned and the performance criteria that was explored .

2 LITERATURE REVIEW

2.1 Basics of PSO

The PSO technique was proposed and initially developed by the electrical engineer Russel C Eberhart and the social psychologist James Kennedy. The technique has a deep connection with some social relations, concepts and behaviours that emerged from a computational study and simulation of a simplified social model of a bird flock seeking for food. This social model belongs to the swarm intelligence. This is an important and extensive research area within natural computing. This algorithm is based on the premise that the knowledge can be gained from social sharing of information among generations and between elements of the same generation [5].

2.2 Inspiration behind PSO

The PSO was built by abstracting the working mechanism of the natural phenomenon through mimicking the navigation pattern and foraging swarms in nature, such as flocks of bird or school of fishes and many other types of nature inspired swarms. PSO idealises the notion such that a group of individuals or particles interconnect, link together,interact and communicate reliably or meaderingly with one another using search direction or gradients. In the algorithm, established particles are used where they owing over a search space for global optima location. Throughout each iteration in the algorithm, the particle will move to a location based on the preceding knowledge or experience as well as the knowledge obtained from the search of the neighbourhood. That means looking at a given particle bird, bat or fish, they maintain position and learn from experience encountered when they navigate as a flock of bats, swarm of birds or school of fishes. This navigation pattern of particles is of importance as communication is vital during this process. Feedback is received from both the local and global best during the global best search process [11].
There have been a number of scientists that have created various computer simulations. These computer simulations take up various interpretations of the movement of organisms in a fish school or bird flock. Notably Reynolds and Heppner and Grenander presented simulations of bird flocking. Reynolds was particularly intrigued by the aesthetics of bird flocking choreography. Heppner was interested in discovering the underlying rules that enabled a large number of birds to flock synchronously, often changing direction suddenly, whilst subsequently scattering and regrouping [6]. Both of the scientists had the insight that local processes such as those modelled by cellular automata, might underlie the unpredictable group dynamics of the social behaviour of a bird. Both of the models,

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

developed by the scientists, relied heavily on manipulation of inter-individual distances. This is the synchrony of flocking behaviour which was thought to be a function of birds' efforts to maintain an optimum distance between themselves and their neighbors [6].

2.3 Advantages of PSO

There are many features of the PSO that make it so efficient in solving optimisation problems. Notably the PSO does not have many parameters. In fact, if the algorithm is compared with other heuristics it will be found that PSO always has less parameters to tune. Furthermore, the PSO's underlying concepts are so simple[12]. Therefore, implementing the algorithm using code can often by straightforward. Also, the algorithm often provides fast convergence which is always a positive attribute to have. Furthermore, the PSO algorithm has less computation burden in comparison with most other heuristics. Whilst the algorithm has less computational burden and fast convergence, it also provides high accuracy in the outputs it produces. It is often the case that when an algorithm has to start from a random point/points in the search space, these starting solutions can have an effect on the outputs of the algorithm[12]. However in the PSO, initial solutions that are generated in the initialisation phase of the algorithm do not have any effect of its computational behaviour. Further to the point, the behaviour of the algorithm is not highly affected by an increase in dimensionality. Finally, there exists many efficient strategies in PSO for mitigating premature convergence, which is why the success rate of the algorithm is so high[12].

2.4 PSO Algorithm

Here is the step by step explanation of the iterative process of the PSO algorithm [14].

(1) Initialise a population of particles. Each particle has to be assigned a random position.
(2) Iterate through each particle and evaluate its fitness by using an optimisation fitness function.
(3) Compare each particle's updated fitness evaluation with its best fitness (pbest). If the updated fitness is better than its pbest then update its best location to the current location and its best fitness to its current fitness.
(4) Find the particle which has the pbest value of all the particles and store its value as the gbest.
(5) Change the velocity and position of the particle according to equations (1) and (2)
(6) Loop to step 2 unless the stop criteria has been met which is usally a sufficiently good fitness or a maximum number of iterations.

2.5 Mathematical Representations of PSO

$$v_{i+1} = v_i + c_1 rand()(p_{best} - x_i) + c_2 rand()(g_{best} - x_i) \quad (1)$$

$$x_{i+1} = x_i + v_i \quad (2)$$

In equation 1 g_{best} is the ith component of the best pointed visited by the neighbours of the particle. x_i is the ith component of the particles current location. p_{best} is the ith component of its personal best. $rand()$ is a independent random variable that is uniformly distributed in $[0, 1]$. c_1 and c_2 are two constants that are referred to as acceleration coefficients which control the relative proportion of cognition and social interaction in the swarm [12]. The equivalent formula is used independently for each dimension on the problem and synchronously for all particles. The position of a particle is updated every time step using equation 2. The next iteration will take place when all particles have been moved. Eventually the whole swarm ill move close to the best location like a flock of birds collectively foraging for food [12].
There are three terms in equation 1. Each of these terms can be linked to one of the theoretical processes in swarm intelligence. Firstly the first time in equation 1 models the tendency of a particle to remain in the same direction it has traversing. This can be referred to as the "inertia", "habit" or "momentum" [13]. The second term in equation 1 is a linear attraction towards the particle's best personal experience scaled by a random weight. This is referred to as "memory","nostalgia" or "self-knowledge" [13]. The third term of equation 1 is the linear attraction towards the best experience of all particles in the swarm. This is also scaled by a random weight. This term is called "cooperation", "shared information" or "social knowledge" [13].

2.6 Travelling Salesman Problem

Literature provides numerous differing methods and approaches in terms of the TSP solutions. From a theoretical point of view, the TSP is seen by some authors as a specific problem within the theory of graphs and networks because visited places can be viewed as certain nodes and the network edges create transport links between them. Literature can also represent the nodes as particular places in a certain geographical territory. They can be perceived as destinations in a map and a contemporary navigation technology can be used for the solution [4].
The travelling salesman problem (TSP) entails that you are given a collection of n cities and the shortest route needs to be found where you start and end at the same city within that route. Given a finite set ofN cities and a distance matrix $(c_{ij})(i, j \epsilon N)$ the following mathematical expression can be used to express the problem [10]:

$$min_\pi \sum_{i \epsilon N} c_{i\pi(i)}, \quad (3)$$

In equation 3 π runs overall cyclic permutations of N, $\pi^k(i)$ is the kth city reached by the salesman from city i. If $N = 1, ..., n$ then the equivalent formulation is also valid[10]:

$$min_v(\sum_{i=1}^{i=n-1} c_{v(i)v(i+1)} + c_{v(n)v(1)} \quad (4)$$

In equation 4, v runs overall permutations of N and v(k) is the kth city in a salesman's tour. Consider G which represents the complete directed graph on the vertex set N with a weight c_{ij} for each arc (i, j). With this holding, an optimal tour corresponds to a Hamiltonian circuit on G of minimum total weight. A Hamiltonian circuit is a circuit passing through each vertex exactly once. If $c_{ij} = c_{ji}$ for all (i, j), then the problem is called symmetric. If this is not the case then the problem will be referred to as asymmetric. If $c_{ik} <= c_{ij} + c_{jk}$ for all (i, j, k), the problem is called euclidean [10].

2.7 Difficulties of TSP

The theoretical solution of the basic model is easy. It is sufficient to explore all possible routes (sequence of visits to particular places) and find out which of them provides the minimum value of the objective function. If we are to visit (n-1) places in sequence, then the total number of all possible routes is given by a permutation of the number $(n-1)!$. For a low value n these routes can be explicitly defined but, when the number of visited places increases the total number of all possible routes rises sharply [8].

Although the problem is easy to state, the travelling salesman problem (TSP) is difficult to solve. The number of possible tours increases exponentially with the number of cities. It is representative of an important class of optimisation problems that are referred to as NP-complete. These problems are interconvertible, but the computing effort needed to solve them increases faster than any power of N. This is because perfect solutions are unattainable for problems of any appreciable size. Also, there has been interest in heuristic methods that find near-optimal solutions in a reasonable time [3].

3 MAIN BODY

This section will now explain how the PSO can be adapted to solve the TSP problem and detail two other algorithms that it can be compared to. This section will also detail various representations of the TSP that the algorithm will use to find optimal solutions for the TSP instances.

3.1 TSP Representations

The most natural representation of a TSP tour is path representation [9]. In path representation, the n cities that should be visited are put in order according to a list of n elements, so that if the city i is the j-th element of list, city i is the jth city to be visited. This representation allows a great number of crossover and mutation operators that have been developed [9].

$$[5, 2, 8, 4, 9, 7] \quad represents \quad 5 \rightarrow 2 \rightarrow 8 \rightarrow 4 \rightarrow 9 \rightarrow 7 \quad (5)$$

So each TSP path/route is a possible solution to a specific problem instance (if all rules are followed). The the sum of the distances between each city is the cost (fitness of solution). Consequently, we are trying to find paths with a lower summed cost as this would represent a shorter route. Therefore we can consider the TSP as a minimisation problem. The distance between each city is calculated by using the euclidean distance [2].

$$d((i,k),(h,k)) = \sqrt{(j-i)^2 + (k-h)^2} \quad (6)$$

To access distances at ease within the PSO algorithm, a distance matrix will be created which will consist of the euclidean distance between each city.

Figure 1: Example of Adjacency Matrix for a TSP Problem Instance [7]

3.2 Random Search

The random search (RS) algorithm starts from a random initial solution. This solution is then mutated to create a mutated path. If this mutated solution has a better fitness than the solution current possessed, then this solution will be replaced by the mutated solution. This process will repeat until the maximum number of iterations is reached.

Algorithm 1 Random Search Algorithm

(1) Initialise Random TSP Path (x)
(2) Calculate cost of Path ($xcost$)
(3) Choose random city in x (v)
(4) Swap city v with next city to create mutated path (mut)
(5) Calculate cost of mutated path ($mutcost$)
(6) If $mutcost < xcost$ then $x = mut$ and $xcost = mutcost$
(7) Check if maximum number of iterations has been reached, otherwise go back to step 3

3.3 Stochastic Hillclimber

The stochastic (SH) algorithm starts from a random initial solution and then looks through the neighbourhood of this solution. In this paper the neighbourhood is all the mutated paths of a path. The fitness is then calculated of each of the solutions in the neighbourhood. The solutions where the fitness has increased are stored in an array. Then one of these solutions are chosen at random. This will be your new solution and this process will repeat until the maximum number of iterations has been reached.

Algorithm 2 Stochastic Hillclimber Algorithm

(1) Initialise Random TSP Path (x)
(2) Calculate cost of Path ($xcost$)
(3) Generate neighbourhood of x using GH Algorithm
(4) Find all neighbours of x that have an improved fitness
(5) Choose a random path from these neighbours (v) and store its cost (vcost)
(6) x = v and xcost = vcost
(7) Check if max number of iterations has been reached, otherwise go to step 3

Algorithm 3 Generate Neighbourhood (GH) Algorithm

```
x is current path
n is number of cities in x
neighbours = []
while i < n do
    new_path = x.copy()
    new_path[i],new_path[i+1] = new_path[i+1],new_path[i]
    Add new_path to neighbours
end while
Return neighbours
```

3.4 Adapted PSO

There exists many variations of the PSO that can be used on the TSP problem. Here is the psuedocode for this papers choice of adapted PSO for a TSP problem. The following algorithm is inspired by [1].

Algorithm 4 Adapted PSO (APSO)

(1) Step 1:
Initialise a swarm of s particles where each particles is assigned a random starting position(x) Each particle's pbest = x
(2) Step 2:
Calculate the fitness of position for each particle.
(3) Step 3:
Using fitness' from Step 2 calculate gbest.
(4) Step 4:
For each particle, calculate the new position:

$$x = pbest \otimes gbest \tag{7}$$

$\otimes$ denotes the heuristic crossover operator
(5) Step 5:
Update pbest If cost of x is less than pbest then pbest = x
(6) Step 6:
Update gbest If cost of pbest is less than gbest then gbest = pbest
(7) Step 7:
Check if stop condition is met. Otherwise go to Step 4.

The algorithms begins by a swarm of s particles being initialised. In this process each particle is assigned a random starting TSP solution. In the next step, the fitness and gbest values are updated according to the pbest value.In the next step, in order to obtain new positions the crossover operator is applied between the gbest and pbest value for each particle. These new positions will be recorded if they take up a lesser distance in comparison to the fitness values of the current position (x) and pbest position. The stopping condition will be met when a specified amount of iterations has been reached. Otherwise the algorithm will continue to calculate new solutions.

3.5 Heuristic Crossover

In step 4 of the adapted PSO, a heuristic crossover (HC) is completed between the pbest and gbest of each particle to identify a new position for the particle to explore. The HC algorithms begins by choosing one of the cities at random (v). Then the two edges emerging from the city in the two routes are compared and the shorter edge is selected. The city on the end of the edge is the new starting city. The process, just mentioned, is repeated again continually until the new route being formed consists of every city. At any point if the shorter edge causes a cycle, then the longer edge will be selected. If the longer edge also causes a cycle then the shortest of randomly generated edges is selected. The following psuedocode explains the algorithm for this process. This was inspired by [1].

Algorithm 5 Heuristic Crossover (HC)

```
Input: Two Solutions x1 and x2
Output: One Solution x
Procedure:
Choose a random city v
Move the city v to the beginning of x1 and x2
Initialize x by v
i=2; j=2;
while (ij) <= n do
    if (x1(i)x2(j)) ∈ x then
        i = i + 1
    end if
    if x1(i) ∈ x then
        Concatenate x2(j) to x
        j = j + 1
    end if
    if x2(j) ∈ x then
        Concatenate x1(i) to x
        i = i + 1
    else
        Let u be the last city in x
        if distance[u,x1(i)]<distance[u,x2(j)] then
            concatenate x1(i) to x
            i = i + 1
        else
            concatenate x2(j) to x
            j = j + 1
        end if
    end if
end while
```

3.6 Algorithm Performance

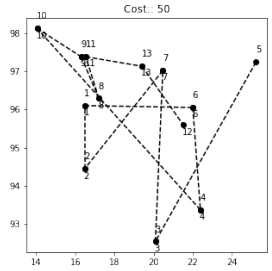

Figure 2: Visualisation of Random Path

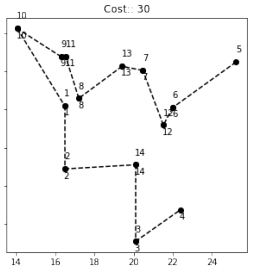

Figure 3: Visualisation of PSO Best Path

In figure 3 you can see the best path found by the PSO algorithm. It can be seen that the cost of the best path found is far better than that of a random path. These solutions are based on the TSP instance burma14. For this instance the cost of the optimal path is actually 30, which shows that on this instance the TSP has found an optimal path.

4 EXPERIMENTATION

There are two main lines of experimentation conducted by this paper. The first line of experimentation is the parameter tuning in the PSO. The second line of experimentation is the algorithmic comparison. This is where the performances of the PSO will be compared to the performances of the RS and SH. In these experiments, the PSO will use the parameters that were chosen based on the parameter tuning experimentation. Throughout all the experimentation mentioned above there are four lines of performance inquiry. The first is the average cost. After the algorithm is implemented 30 times on a TSP instance, the average of all the final costs will be calculated to assess the quality of the solutions outputted from algorithms. Another line of inquiry is the average clock time, which is calculated by taking the mean average of all all the clock times from all the iterations. Another line of inquiry is the average relative error. In each iteration, an error percentage is calculated using the optimal cost and the final cost of the algorithm. The mean average is taken of all these error percentages. The final line of inquiry is the best solution, which is the solution with least cost found throughout all the iterations.

4.1 Problem Instances

There are five TSP instances being used where the algorithms and parameters will be explored on. These TSP instances have different variations in the amount of cities. The largest amount of cities in these instances is 99. Whilst this limits the performance tests of the algorithms, due to time constraints and processor limitations this is the greatest amount of cities that can be used in a particular instance. The problem instances used are berlin52, eil76, burma14, dantzig42 and rat99 where the number in the name specifies the number of cities in the instance. Each city has a position (x,y) assigned to it.

4.2 Parameter Tuning

In this section,the performance of the PSO with different parameter values is explored. The characteristics of performance are analysed with a view to these parameters being used in experiments where the algorithm has to be implemented 30 times on an instance to account for stochasticity. Each of the experiments are implemented on the dantzig42 instance.
The first parameter that is explored, is the number of particles that are initialised at the start of the algorithm.

Amount of Particles	Best Cost	Average Cost	Average Clock Time	Average Relative Error
100	756.9	867.0	1.8	0.24
250	717.8	820.3	3.8	0.17
500	707.9	796.7	7.6	0.14
750	706.4	783.4	11.7	0.12
1000	689.0	789.9	15.5	0.13

Figure 4: Results of varying the Amount of Particles

There are five different amount of particles used. Theoretically, more amounts of particles could have been explored as it would have enabled the PSO to find more optimal solutions. However due to the large compuation times some particle amounts created this were the suitable particle amounts that could be explored.

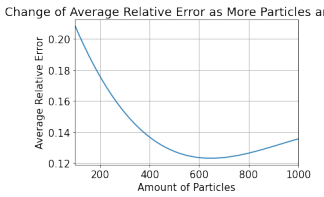

Figure 5: Progression of Relative Error

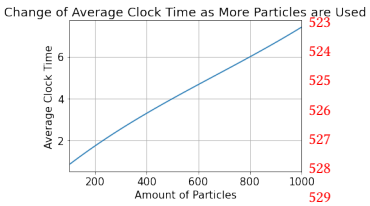

Figure 6: Progression of Clock Time

In figure 5 you can see that the average relative error decreases as more particles are used up until 600 particles. The progression of average cost also follows a similar pattern. After this point, the cost starts to increase again, which is a surprise as it would be expected if there were more new positions being calculated a higher chance of finding better solutions would exist. Whilst this is a major positive, the major drawback of more particles being used is seen in figure 6. The average clock time rapidly increases as the amount of particles increases. It can be seen that the increase in clock time, is actually quite rapid. Considering how many iterations an algorithm would have to be implemented, this means extremely large computation times may be required when the amount of particles is too high. Furthermore, if you assess 5 it can be seen that the relative error has only increased by 0.08 across the iteration increase, whilst the clock time increases by nearly 8 seconds. Therefore there is not an adequate increase of quality of solutions despite a huge increase in computational time. In fact a relative error of 0.18 is not too large, so the amount of particles chosen is 100 particles.

The other parameter we are exploring is the amount of iterations the algorithm will stop after.

Amount of Iterations	Best Cost	Average Cost	Average Clock Time	Average Relative Error
20	917.6	1083.2	0.3	0.55
50	723.8	864.1	0.8	0.23
100	734.0	822.2	1.5	0.18
150	728.6	844.1	2.3	0.20
200	722.6	822.7	3.0	0.17

Figure 7: Results of varying the Amount of Iterations

Here there are five amounts of iterations being explored. There could have been other iteration amounts explored, however due to the large computation times of each PSO experiment these were the only iteration amounts that were explored.

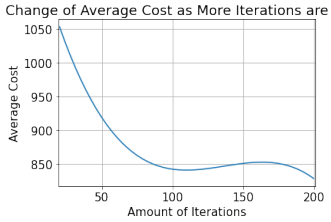

Figure 8: Progression of Relative Error

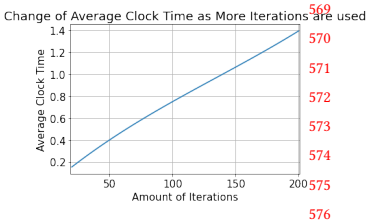

Figure 9: Progression of Clock Time

It can be seen that the progression of the average cost as the amount of iterations does not takes up a constant increase or decrease. The progression of the average relative error also takes a similar progression. It can be seen that the average cost rapidly decreases up until 80 iterations, increases again until 175 iterations before decreasing again. Investigating larger amounts of iterations would enable us to find out about the extent of this decrease, however with the data we have the optimal number of iterations is around 100. This is surprising as more iterations would mean that more positions are being calculated so better solutions could be found. It can also be seen that the average clock time rapidly increases when the amount of iterations are increased, similar to the progression when the amount of particles are increased. Since an optimal solution is found at 100 iterations, the choice of iterations should be between 0 to 100. Analysing the tradeoff, it can be seen whilst there is an increase of 1.5s in average clock time in this range, there is a decrease of almost 300 in terms of average cost. This paper considers this computational time worthy of the quality of solution gained, so the amount of iterations the algorithms will stop at is 100 iterations.

4.3 Algorithmic Comparison

In this section the performance of the PSO is contextualised by comparing its performance to the stochastic hillclimbing and random search algorithms. To complete this experiment, each of these algorithms was implement on each of the TSP instances 30 times. The PSO uses the parameters chosen after the parameter tuning section.

Random Search Results	Best Cost	Average Cost	Average Clock Time	Average Relative Error	Best Known Cost
Berlin52	21049.1	24641.2	0	2.3	7542
Eil76	1975.8	2140.3	0	3.0	538
Burma14	41.0	49.6	0	0.7	30
Dantzig42	1985.5	2366.0	0	2.4	699
Rat99	6352.1	7132.6	0	4.9	1211

Figure 10: Results of the Random Search Algorithm

Stochastic Hillclimber Results	Best Cost	Average Cost	Average Clock Time	Average Relative Error	Best Known Cost
Berlin52	20446.9	20662.2	1.9	1.7	7542
Eil76	1804.4	1844.5	4.4	2.4	538
Burma14	43.2	43.2	0	0.4	30
Dantzig42	2002.8	2006.9	0.9	1.9	699
Rat99	5849.8	6291.2	10.7	4.2	1211

Figure 11: Results of the Stochastic Hillclimber

PSO Results	Best Cost	Average Cost	Average Clock Time	Average Relative Error	Best Known Cost
Berlin52	10101.1	10101.1	14.8	0.3	7542
Eil76	711.4	711.4	19.2	0.3	538
Burma14	34.4	34.4	4.4	0.1	30
Dantzig42	734.6	734.6	9.3	0.1	699
Rat99	1711.7	1717.8	33.7	0.4	1211

Figure 12: Results of the PSO

You will be able to see in figure 13 that the PSO algorithm has the lowest average cost in each instance. Also by assessing figures you will be able to see that this is the case in terms of relative error and best solution found. This shows that in terms of finding solutions close to the optimal, the PSO is a far better algorithm than random search and the stochastic hillclimber.

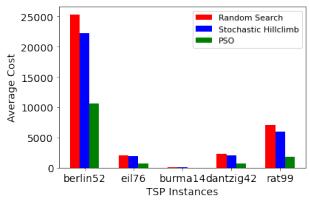

Figure 13: Progression of Relative Error

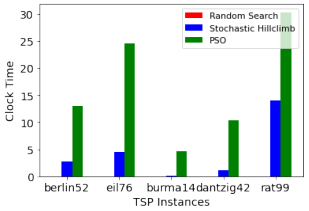

Figure 14: Progression of Clock Time

However in figure 14 it can be seen that the PSO algorithm has larger average computational times in comparison to the other algorithms. Note that there are no bars that can be seen for random search as the algorithm is so quick. Assessing these results there is a clear tradeoff between computational time and solution quality, however it is difficult to assert whether the extra computational time is worth the solution quality.

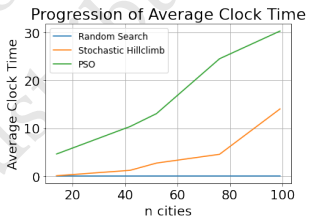

Figure 15: Progression of Relative Error

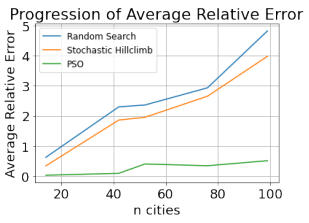

Figure 16: Progression of Clock Time

It can be seen in figure 15 that rate that the PSO potentially increases in computational time actually increases, as the amount of cities increases. Therefore if TSP instances where there is a greater number of cities than 99 are used, the PSO's computation time could be exponentially greater than that of random search and stochastic hillclimb. However within the range of cities the paper has analysed it can be seen that the PSO is roughly twice as long as the stochastic hillclimber throughout their respective progressions. If figure 16 it can be seen that the PSO is consistently finding greater solutions in comparison to the other algorithms as the amount of cities increases. In the range of cities analysed, the PSO is finding a solutions up to ten times greater than the random search. Therefore considering the added computation time, you are rewarded with a huge increase in quality of solutions. Therefore within the instances analysed the PSO is best trade off between computation time and quality of solution. However there is evidence to suggest that this may not be the case if instances with greater amounts of cities are considered as the computational time may start getting extremely large.

REFERENCES

[1] Keivan Borna and Razieh Khezri. 2015. A combination of genetic algorithm and particle swarm optimization method for solving traveling salesman problem. *Cogent Mathematics* 2, 1 (2015), 1048581.

[2] Per-Erik Danielsson. 1980. Euclidean distance mapping. *Computer Graphics and image processing* 14, 3 (1980), 227–248.

[3] Richard Durbin and David Willshaw. 1987. An analogue approach to the travelling salesman problem using an elastic net method. *Nature* 326, 6114 (1987), 689–691.

[4] Exnar Filip and Machac Otakar. 2011. The travelling salesman problem and its application in logistic practice. *WSEAS Transactions on Business and Economics* 8, 4 (2011), 163–173.

[5] Diogo Freitas, Luiz Guerreiro Lopes, and Fernando Morgado-Dias. 2020. Particle swarm optimisation: a historical review up to the current developments. *Entropy* 22, 3 (2020), 362.

[6] James Kennedy and Russell Eberhart. 1995. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, Vol. 4. IEEE, 1942–1948.

[7] I-Hong Kuo, Shi-Jinn Horng, Tzong-Wann Kao, Tsung-Lieh Lin, Cheng-Ling Lee, Yuan-Hsin Chen, YI Pan, and Takao Terano. 2010. A hybrid swarm intelligence algorithm for the travelling salesman problem. *Expert systems* 27, 3 (2010), 166–179.

[8] Gilbert Laporte and Udatta Palekar. 2002. Some applications of the clustered travelling salesman problem. *Journal of the operational Research Society* 53, 9 (2002), 972–976.

[9] Pedro Larranaga, Cindy M. H. Kuijpers, Roberto H. Murga, Inaki Inza, and Sejla Dizdarevic. 1999. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial intelligence review* 13 (1999), 129–170.

[10] Jan Karel Lenstra and AHG Rinnooy Kan. 1975. Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society* 26, 4 (1975), 717–733.

[11] Modestus O Okwu and Lagouge K Tartibu. 2020. *Metaheuristic optimization: Nature-inspired algorithms swarm and computational intelligence, theory and applications*. Vol. 927. Springer Nature.

[12] Riccardo Poli. 2008. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications* 2008 (2008), 1–10.

[13] Ahmad Rezaee Jordehi and Jasronita Jasni. 2015. Particle swarm optimisation for discrete optimisation problems: a review. *Artificial Intelligence Review* 43 (2015), 243–258.

[14] Yuhui Shi. 2004. Particle swarm optimization. *IEEE connections* 2, 1 (2004), 8–13.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009