Overview of different Particle Swarm Optimisation methods on the Travelling Salesman Problem

Sushen Yadav

ABSTRACT

This report contains the analysis of running 8 computational experiments with multiple PSO methods. The primary goal defining the design of these experiments is to analyse the effect of various local heuristic methods on the best solution found by PSO across TSP instances. We define six different PSO algorithms from the literature and implement them in python. Each PSO algorithm incorporates a specially designed local heuristic. We consider 3 TSP instances, from a small size of 29 to 280 and finally 1400 nodes. The TSP instance [7] is saved as a distance matrix for faster distance calculation. We have ensured that the TSP rules of visiting each node once and inclusion of distance between last visited node and origin node is strictly followed. The distance calculation method used is the euclidean distance between two nodes. We use pylint for identifying linting errors, Multiprocessing for speeding up the execution time and Object-oriented programming for organisation of our code. Each algorithm is pitted against each other and random sampling, Stochastic hill climber. The experiments are designed to help understand the behaviour of the hyperparameters and identify an optimal value from a set of 4 values. We conclude with an observation about the hyperparameters and further insight into the operation of these particle swarm optimisations.

KEYWORDS

Particle swarm optimisation, Travelling salesman problem

ACM Reference Format:
Sushen Yadav. 2024. Overview of different Particle Swarm Optimisation methods on the Travelling Salesman Problem . In *Proceedings of none (Evolutionary Computing CA2).* , 10 pages.

1 INTRODUCTION

PSO techniques have received a lot of recognition in the Optimisation community. This has led to the spawn of many similar approaches which modify the PSO ever so slightly to get a newer method. A common theme is to add a local heuristic to PSO to improve it's performance. To gain insight into enhancing PSO performance for TSP, we employed five diverse PSO algorithms, each incorporating a distinct local heuristic method for adjusting particle positions and velocities. These methods were chosen from a multitude of PSO methods as each of them incorporate a unique local heuristic. This will make sure that the report encompasses a broad range of literature, rather than focusing on one particular area.

2 LITERATURE REVIEW

The Traveling Salesman Problem (TSP) [8] is a well-known combinatorial optimization problem. The problem involves finding the shortest possible route that visits each city exactly once and returns to the starting city, given a list of cities and the distances between each pair of cities. The number of possible routes increase exponentially with increase in nodes. For example there are 300,000 routes with 10 nodes and 10^{18} with 20 nodes. This makes it impractical to solve TSP optimally by exhaustively checking all possible routes. No one has found a polynomial-time algorithm that can solve TSP optimally for all instances. The best-known exact algorithms have exponential worst-case time complexity. Even finding a good approximation to the optimal TSP solution is difficult. It has been proven that if $P \neq NP$, then there is no polynomial-time approximation scheme for TSP that can approximate the optimal solution within any constant factor.

The Particle Swarm Optimization (PSO) [5] The functioning of the Particle Swarm Optimization (PSO) algorithm is shaped by a variety of control parameters, such as the problem dimension (n), the number of particles (n_s), acceleration coefficients (c_1 and c_2), inertia weight (w), neighborhood size (n_N), and random values (r_1 and r_2) that modulate the cognitive and social components. The initial diversity and the per-iteration computational complexity are dependent on the swarm size. Based on empirical data, it is generally recommended to work with 10 to 30 particles [11][1], although the optimal size may differ according to the specific problem. Smaller neighborhoods lead to slower but more reliable convergence and are less susceptible to local minima. The number of iterations required for a good solution is also problem-dependent. The acceleration coefficients c_1 and c_2, along with the random vectors r_1 and r_2, control the stochastic influence of the cognitive and social components on a particle's velocity. When both c_1 and c_2 are zero, particles will continue moving at their current speeds until they reach a boundary. If c_1 is less than zero and c_2 is zero, the particles act as independent hill-climbers. On the other hand, if c_2 is less than zero and c_1 is zero, the swarm is drawn towards a single point, behaving like a stochastic hill-climber. Particles are most effective when $c_1 \approx c_2$, balancing the cognitive and social components. The ratio between c_1 and c_2 may be problem-dependent with plain search areas having more chance to utilize a larger social component while potentially rough, multi-modal search spaces will be benefited from a larger cognitive component. Proper initialization of c_1 and c_2 is of great importance to prevent divergence or cyclic behaviour.

3 ALGORITHMS

The following section contains definition of the algorithms that were considered. The pseudocode for each implementation is added to appendix B.

3.1 APSO

The inertia weight w in PSO controls the influence of a particle's previous velocity on its current velocity. It is the primary agent balancing global exploration and local exploitation. Ahmad et al. [6]

Evolutionary Computing CA2, May 01, 2024, University of Exeter,UK
2024.

proposed an adaptive inertia weight scheme based on the evolutionary state estimation (ESE) technique. This technique assesses the evolutionary state of the algorithm using population distribution information. The adaptive inertia weight is defined as:

$$w_i(t) = w_0 - \frac{w_0}{1 + e^{-m_i(t)}}$$

where w_0 is the initial inertia weight, and $m_i(t)$ represents the slope of the objective function at the current point, estimated as:

$$m_i(t) = \frac{f_{it}(x_i(t)) - f_{it}(x_i(t-1))}{||x_i(t) - x_i(t-1)||}$$

The adaptive inertia weight $w_i(t)$ decreases as the slope $m_i(t)$ increases. A higher slope indicates that the particle is located in a more promising region while a smaller inertia weight is used to enhance local exploitation. Thus, a lower slope suggests that the particle is in a less promising area and a larger inertia weight should be employed to promote global exploration.

3.2 HPSO

The binary Particle Swarm Optimization [4] has particles that represent positions in a binary search space, where each element of particle's position vector assumes the value of either 0 or 1. Velocity is defined as the probability of a bit being in one state or the other. A tanh function is employed to scale velocity rangers to $[0, 1]$. The update position formula thus becomes $x_{ij}(t+1) = 1$ if $r_{3j}(t) < \text{sig}(v_{ij}(t+1))$, and 0 otherwise, where $r_{3j}(t) \sim U(0, 1)$. The velocity calculation equation has the same form as that in the PSO, with $x_i, y_i, \hat{y} \in \mathbb{B}^{n_x}$ and $v_i \in \mathbb{R}^{n_x}$.

3.3 SPSO

Spatial PSO [10] adds a neighborhood based on the Euclidean distance between particles rather than particle indices. The calculation of these spatial neighborhoods requires computing the Euclidean distance between all particles at each iteration, significantly increasing the computational complexity by $O(n_t n_s^2)$, where n_t is the number of iterations and n_s is the swarm size. The distance-based neighborhoods have the advantage of changing dynamically with each iteration.

3.4 DEPSO

The differential evolution (DE) [3] based PSO incorporates DE's arithmetic crossover operator which involves three randomly selected parents into the PSO algorithm. Each particle dimension is updated as

$$x'_{ij}(t+1) = \begin{cases} x_{1j}(t) + \beta(x_{2j}(t) - x_{3j}(t)) & \text{if } U(0,1) \leq P_c \text{ or } j = U(1, n_x) \\ x_{ij}(t) & \text{otherwise} \end{cases} \tag{1}$$

where $P_c \in (0, 1)$ is the crossover probability and $\beta > 0$ is a scaling factor. The particle's position is only replaced if the offspring has better fitness. Further approaches in the literature[3] include executing the DE process at specified intervals or applying it to each particle for a number of iterations.

3.5 PPPSO

The predator-prey PSO [9] introduces a second swarm of predator particles that pursue the global best prey particle. The predator's velocity update is defined as

$$v_p(t+1) = r(\hat{y}(t) - x_p(t)),$$

where v_p and x_p are the velocity and position vectors of the predator particle, and $r \sim U(0, V_{\max,p})^{n_x}$.

The prey particles' velocity update includes an additional component representing the predator's influence, defined as $c_3 r_{3j}(t) D(d)$, where $D(d) = \alpha e^{-\beta d}$ quantifies the predator's influence based on the Euclidean distance d between the prey and predator. The predator's influence grows exponentially with proximity causing prey particles to scatter when the predator is close. This facilitates exploration of the search space.

3.6 Random Sampling

The Random Sampling approach is essentially a simple method which generates a specified number of random solutions and returns the best solution found among them. It relies on generating random permutations and evaluating their fitness to find the best solution.

3.7 Stochastic Hill Climber

The stochastic hill climbing algorithm uses a local search approach based on the concept of neighborhood. It starts with a random solution and iteratively tries to improve it by swapping two cities in the solution. If the new solution is better, it becomes the new best solution. The algorithm stops when the maximum number of iterations is reached or when the solution does not improve for a specified number of iterations.

The neighborhood $N(s)$ of a solution s is defined as the set of solutions that can be obtained by applying a specific operation (in this case, swapping two cities) to s.

let s be the current solution and s' be a solution in the neighborhood of s, i.e., $s' \in N(s)$. The algorithm moves from s to s' if the fitness of s' is better than the fitness of s:

$$f(s') < f(s)$$

where $f(\cdot)$ represents the fitness function (total distance) of a solution.

The algorithm continues the search process until the maximum number of iterations is reached or until there is no improvement in the best solution for a set number iterations.

4 EXPERIMENTS

All hyperparameters except ablated parameter are set constant. The ablated parameter values are passed as a list to the run-ablated-parameter-experiments function. The Goal is to find set of optimal ablated parameter out of the 4 different choices. The experiment can run on 'n' ablated parameters; However, Computational costs limit experimentation on larger set of values. The plots for each experiment for all the TSP instances are located at appendix A.

4.1 Population size

The population size experiments investigate the impact of swarm size on the performance of PSO algorithms. By varying the population size and measuring the best tour length and the runtime we try to find the optimal swarm size for each algorithm can be determined. The results are visualized using line plots to identify the trends and trade-offs between solution quality and computational efficiency.

The experimental results for population size provide valuable insights that expand upon the information presented in the literature [2]. The literature [2]mentions that the optimal swarm size is problem-dependent and that empirical studies have shown success with small swarm sizes of 10 to 30 particles. The experimental results rightly demonstrate that the optimal population size varies significantly across different algorithms. For PSO and DEPSO, a moderate population size of 60 is found to be optimal. This suggests that a balance between exploration and computational cost is achieved at this size. On the other hand, algorithms like APSO, HPSO, SPSO, and PPPSO benefit from larger population sizes of 100. Thus, the increased diversity and exploration capabilities outweigh the computational overhead of a large population size for these algorithms. Interestingly, simpler algorithms like Random Sampling and Stochastic Hill Climber achieve their best performance with a smaller population size of 20, This proves that they rely more on randomness and local search rather than population dynamics. These findings highlight the importance of considering the specific characteristics and requirements of each algorithm when choosing the optimal population size rather than solely relying on general heuristics.

4.2 Inertia Weight

Inertia weight experiments explore the influence of the inertia weight parameter on the convergence behavior of PSO algorithms. 4 Different inertia weight values are tested, and the best tour lengths are plotted against the corresponding inertia weights.

The experimental results align with the insights provided in the literature [2] regarding the inertia weight hyperparameter. The literature[2] mentions that a high inertia weight can lead to velocity values becoming too large too quickly. This means that a well-thought-out inertia weight can considerably increase the performance of the PSO algorithm. The experimental results show that the Particle Swarm Optimization (PSO) and Discrete PSO (HPSO) achieve optimal performance with an inertia weight of 1.0. On the other hand, the Spatial PSO (SPSO) benefits from a lower inertia weight of 0.4.

4.3 Acceleration Coefficients

Acceleration coefficients experiments examine the effect of cognitive and social learning factors on the search dynamics of PSO algorithms. The findings provide insights into the optimal balance between individual and social learning.

The experimental results for the acceleration coefficients are consistent with the information provided in the literature [2]. The literature[2] discusses the importance of the acceleration coefficients, c1 and c2, in controlling the stochastic influence of the cognitive and social components on the particle's velocity. It mentions that particles are most effective when nostalgia (c1) and envy (c2) coexist in a good balance, i.e., $c1 \simeq c2$. The experimental results show that for algorithms like PSO and HPSO, the optimal acceleration coefficients are 2.0 for the cognitive component and 1.5 for the social component. This combination is a balance between the particle's individual experience and the influence of the global best solution. The optimal acceleration coefficients are 2.0 for both the cognitive and social components of SPSO. This highlights that the equal influence of individual and neighborhood best position leads to optimal solutions for SPSO.

4.4 $Wmin$ and $Wmax$

The $Wmin$ and $Wmax$ experiment investigates the impact of adaptive inertia weight on the performance of the APSO algorithm. The results shed light on the optimal range of inertia weights that allows the APSO algorithm to adapt its search behavior dynamically.

The experimental results for the Adaptive PSO (APSO) corroborate the concepts discussed in the literature[2] regarding the adaptive inertia weight mechanism. The literature[2] highlights the importance of balancing exploration and exploitation throughout the search process. The experimental results show that APSO achieves the best performance with a minimum inertia weight $Wmin$ of 0.2 and a maximum inertia weight $Wmax$ of 0.8. This wide range of inertia weight values enables APSO to adapt its search behavior based on the progress at that iteration. This promotes exploration in the early stages and encourages exploitation and convergence towards the end of the search. The optimal $Wmin$ and $Wmax$ combination aligns with the literature's[2] emphasis on striking a balance between global exploration and local refinement.

4.5 Neighbourhood size

Neighborhood size experiments assess the influence of the neighborhood topology on the performance of the SPSO algorithm.

The experimental results for the Spatial PSO (SPSO) demonstrate the impact of neighborhood size on the algorithm's performance which is inline with the literature[2]. The literature[2] mentions that the neighborhood size defines the extent of social interaction within the swarm. The experimental results show that for larger TSP instances, a smaller neighborhood size of 3 proves to be effective. As this allows for more focused local search and exploitation of promising regions while reducing computational overhead. In contrast, for smaller TSP instances, a larger neighborhood size of 9 is beneficial. This allows SPSO to explore a wider range of solutions and gather more information from neighboring particles. These findings align with the literature's[2] discussion on the adaptive choice of neighborhood size based on the instance size to strike a balance between local exploitation and global exploration.

4.6 Crossover Rate and Mutation Rate

Crossover rate and mutation rate experiments investigate the impact of these parameters on the performance of the DEPSO algorithm.

The experimental results for the Differential Evolution PSO (DEPSO) support the concepts presented in the literature[2] regarding the crossover and mutation rates. The literature[2] emphasizes

the importance of balancing exploration and exploitation in the algorithm. The experimental results show that the optimal crossover rate (CR) of 0.8 and mutation rate (F) of 0.2 achieve this balance in DEPSO. The high CR value ensures the preservation of good solution components, while the low F value introduces a moderate level of mutation. This combination of CR and F values aligns with the book's discussion on effectively navigating the TSP solution space by striking a balance between inheriting good solution components and introducing diversity through mutation.

4.7 Fear Factor

Fear factor experiments explore the influence of the fear factor parameter on the behavior of the predator-prey PSO (PPPSO) algorithm. The experiment tries to find the optimal fear factor range that balances the attraction and repulsion forces between particles and the predator.

The experimental results for the Predator-Prey PSO (PPPSO) confirm the significance of the fear factor parameter as stated by the literature[2]. The literature[2] mentions that the fear factor controls the influence of the predator on the movement of the prey particles. The experimental results show that the optimal fear factor value depends on the size of the TSP instance. For smaller instances, a lower fear factor of 0.2 allows the prey particles to focus more on local search and exploitation, enabling quick convergence towards high-quality solutions. In contrast, for larger instances, a higher fear factor of 0.4 promotes exploration and helps particles escape from local optima. These findings align with the literature's[2] discussion on the adaptive setting of the fear factor based on the instance size to strike a balance between exploration and exploitation.

4.8 Maximum Iterations

Maximum iterations experiments examine the impact of the termination criterion on the performance of PSO algorithms. The results provide insights into the trade-off between solution quality and computational time. Highlighting importance of the selection of an appropriate termination criterion that ensures convergence to high-quality solutions while maintaining computational efficiency.

The experimental results regarding the maximum iterations provide insights that complement the information in the literature[2]. The literature [2] mentions that the number of iterations required to reach a good solution is problem-dependent and that too few iterations may terminate the search prematurely, while too many iterations can lead to unnecessary computational complexity. The experimental results show that the impact of increasing the maximum iterations varies depending on the algorithm and the size of the TSP instance. For PSO, APSO, and HPSO, increasing the maximum iterations may lead to improvement in solution quality for larger instances, as it provides more opportunities for exploration and refinement. However, for smaller instances, increasing the maximum iterations may not result in significant improvement, as the algorithms might have already converged to near-optimal solutions within a smaller number of iterations. These findings highlight the importance of carefully selecting the maximum iterations based on the problem characteristics and the algorithm's behavior.

5 CONCLUSION

The insights into the behavior and effectiveness of each algorithm observed from the experiments has been in accordance with the literature. This report encompassed 6 different style of PSO and analysed the behaviour of all of their hyperparameters.

Even though the results were concurrent with the literature[2], there are important details that should be addressed as they can interfere with the validity of the experiment.

The time complexity analysis is highly specific to this implementation of PSO, a new implementation with a specially designed data structure for tsp for eg: splay trees can yield different results.

We have evaluated 4 options for each hyperparameter based on intuition about the hyperparameter behaviour. However, Hyperparameter optimisation in itself is a growing field where the parameter search space is large.

The computational cost has been prioritised when setting up these experiments. A more comprehensive study would require more wall-clock time as well as compute time.

REFERENCES

[1] R. Brits, A.P. Engelbrecht, and F. van den Bergh. 2002. A Niching Particle Swarm Optimizer. In *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning*. 692–696.

[2] Andries P. Engelbrecht. 2007. *Computational Intelligence: An Introduction*. John Wiley & Sons. 312–342 pages.

[3] T. Hendtlass. 2001. A Combined Swarm Differential Evolution Algorithm for Optimization Problems. In *Proceedings of the Fourteenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Vol. 2070. Springer-Verlag, 11–18.

[4] J. Kennedy and R.C. Eberhart. 1997. A Discrete Binary Version of the Particle Swarm Algorithm. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*. 4104–4109.

[5] J. Kennedy, R.C. Eberhart, and Y. Shi. 2001. *Swarm Intelligence*. Morgan Kaufmann.

[6] Ahmad Nickabadi, Mohammad Mehdi Ebadzadeh, and Reza Safabakhsh. 2011. A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied Soft Computing* 11, 4 (2011), 3658–3670. https://doi.org/10.1016/j.asoc.2011.01.037

[7] Gerhard Reinelt. 1991. TSPLIB–A Traveling Salesman Problem Library. *ORSA Journal on Computing* 3, 4 (1991), 376–384.

[8] Gerhard Reinelt. 1994. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, Berlin, Heidelberg.

[9] A. Silva, A. Neves, and E. Costa. 2002. An Empirical Comparison of Particle Swarm and Predator Prey Optimisation. In *Proceedings of the Thirteenth Irish Conference on Artificial Intelligence and Cognitive Science*, Vol. 2464. Springer-Verlag, 103–110.

[10] P.N. Suganthan. 1999. Particle Swarm Optimiser with Neighborhood Operator. In *Proceedings of the IEEE Congress on Evolutionary Computation*. 1958–1962.

[11] F. van den Bergh and A.P. Engelbrecht. 2001. Effects of Swarm Size on Cooperative Particle Swarm Optimisers. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 892–899.

A PLOTS

B PSEUDO CODE

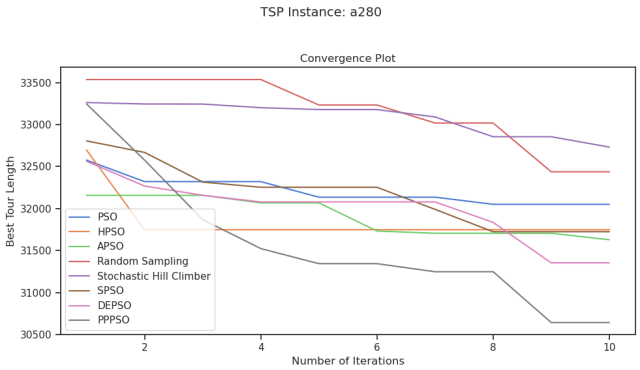

Figure 1: Preliminary Findings a280

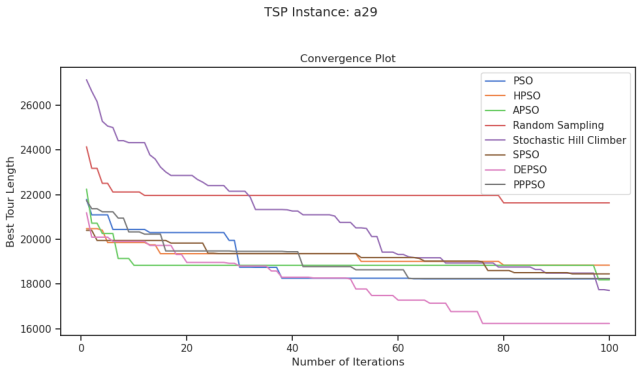

Figure 2: Preliminary Findings a29

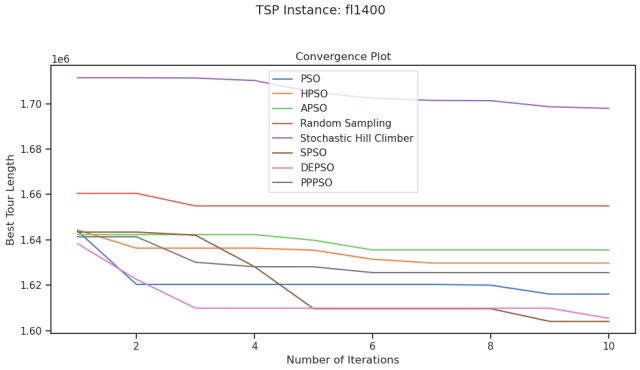

Figure 3: Preliminary Finding fl1400

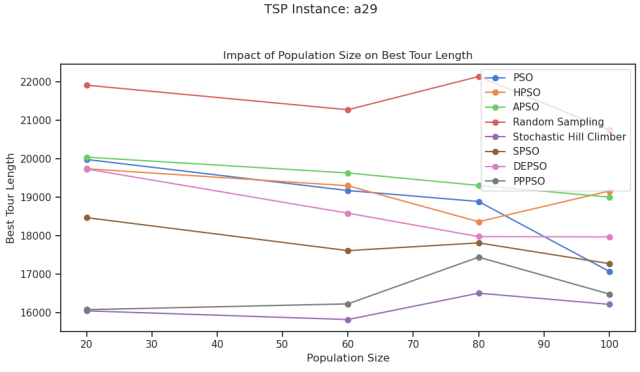

Figure 4: Population Size a29

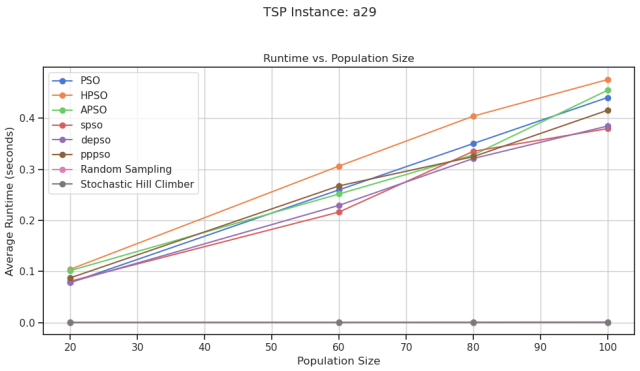

Figure 5: Runtime a29

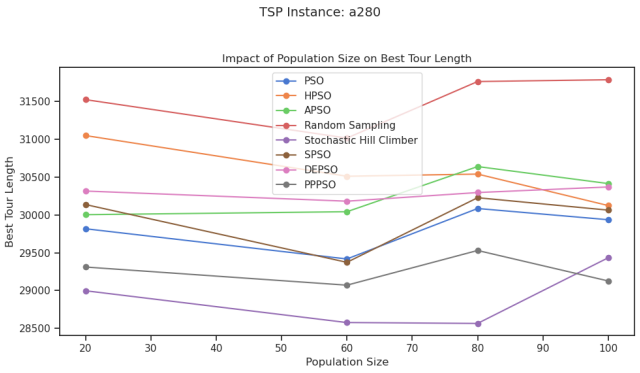

Figure 6: Population Size a280

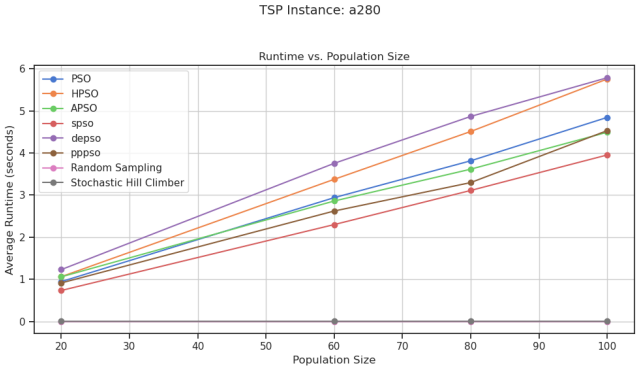

Figure 7: Runtime a280

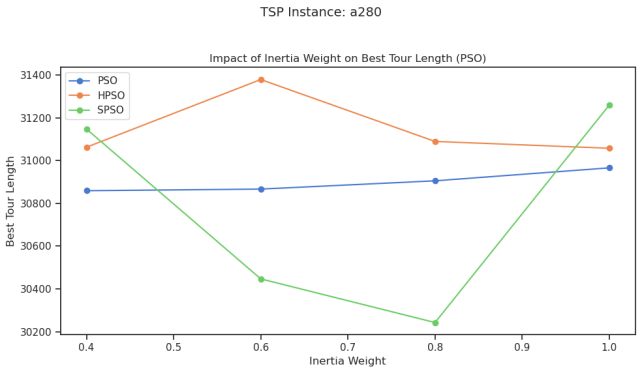

Figure 11: Inertia Weight a280

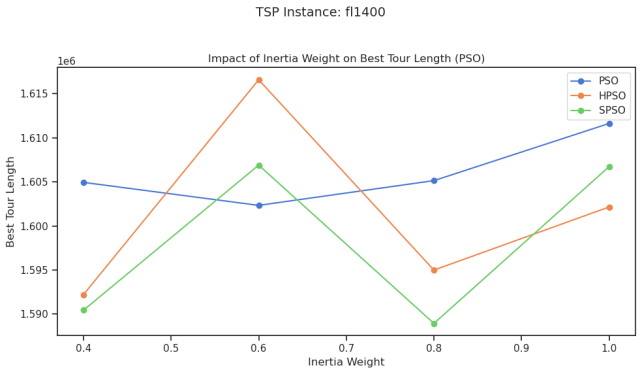

Figure 12: Inertia Weight fl1400

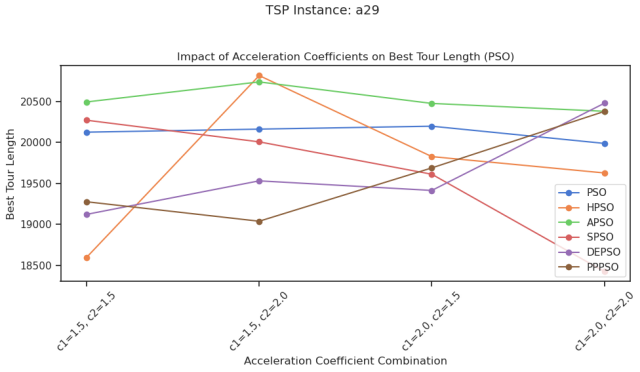

Figure 13: Acceleration Coefficient a29

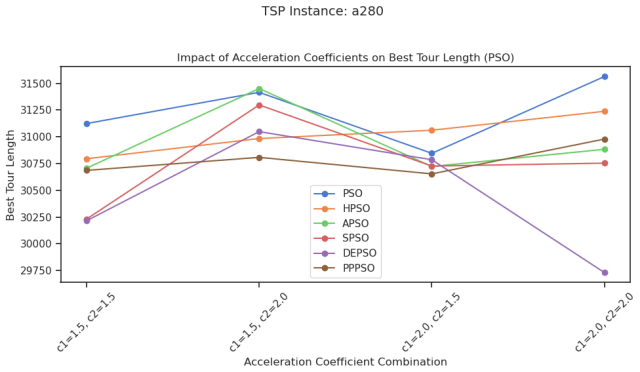

Figure 14: Acceleration Coefficient a280

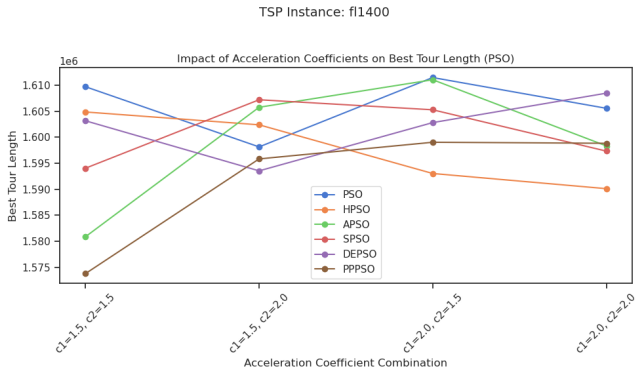

Figure 15: Acceleration Coefficient fl1400

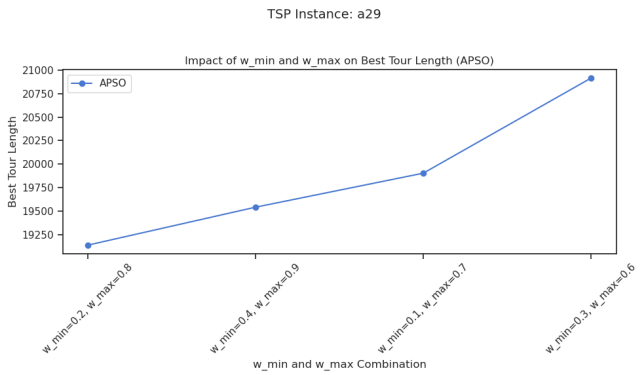

Figure 16: $Wmin$,$Wmax$ a29

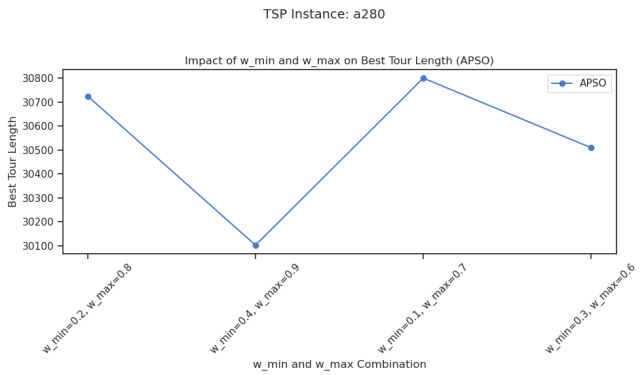

Figure 17: $Wmin$,$Wmax$ a280

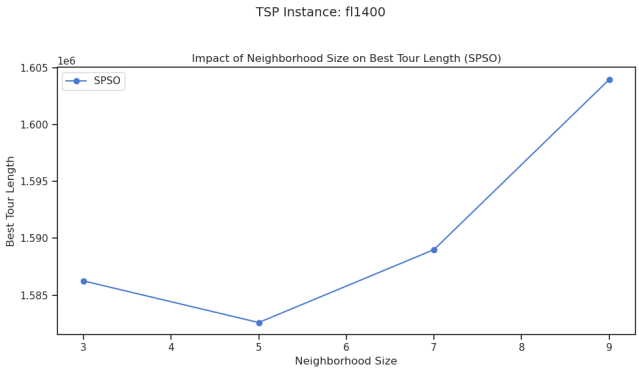

Figure 21: Neighbourhood Size fl1400

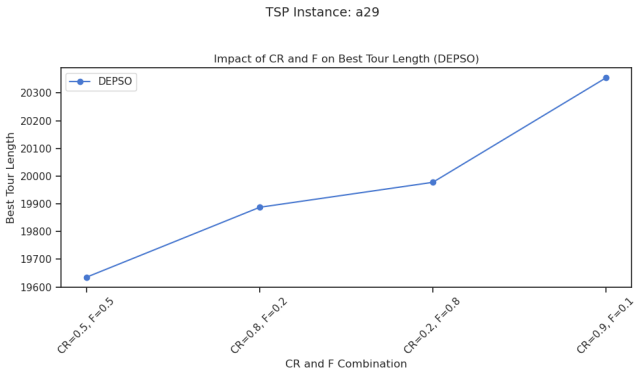

Figure 22: Crossover and Mutation rate a29

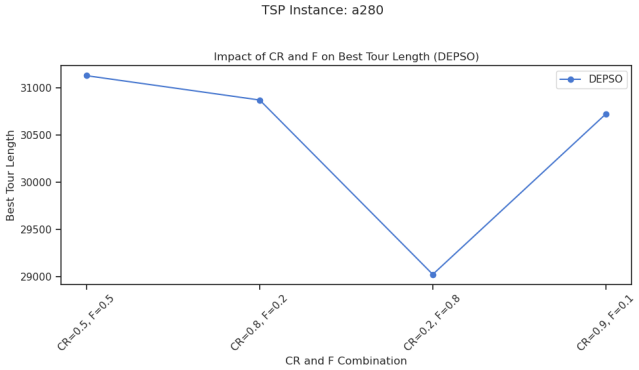

Figure 23: Crossover and Mutation rate a280

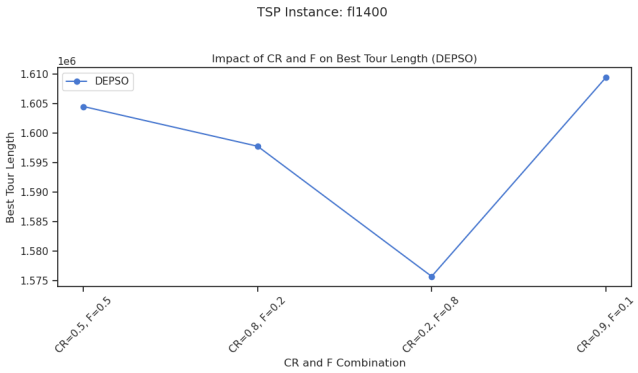

Figure 24: Crossover and Mutation rate fl1400

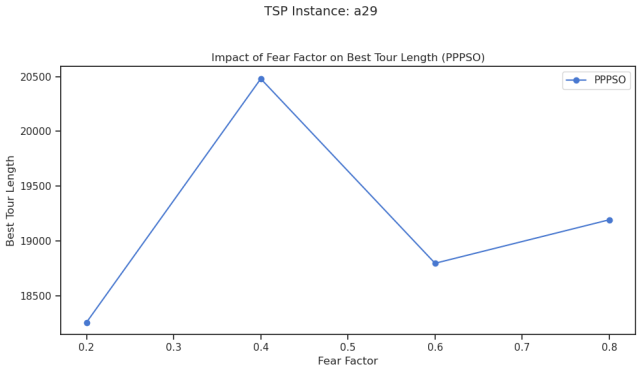

Figure 25: Fear Factor a29

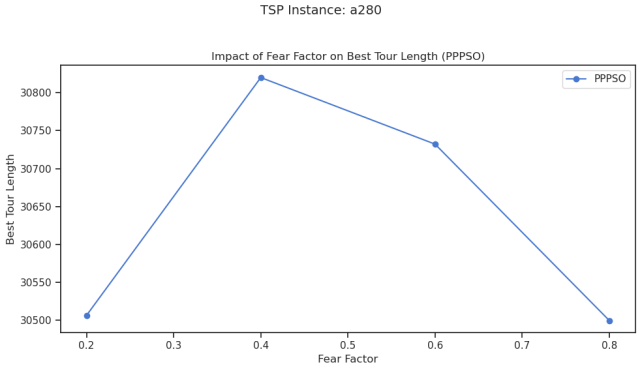

Figure 26: Fear Factor a280

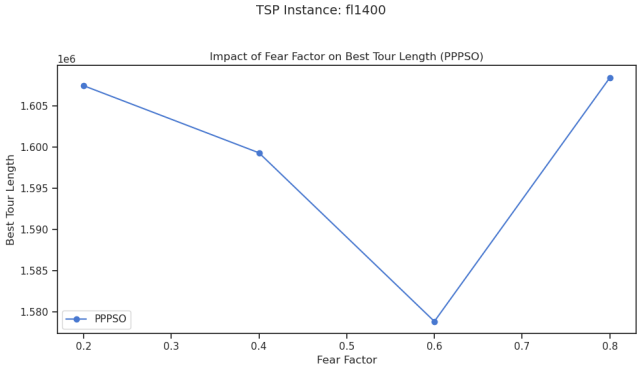

Figure 27: Fear Factor fl1400

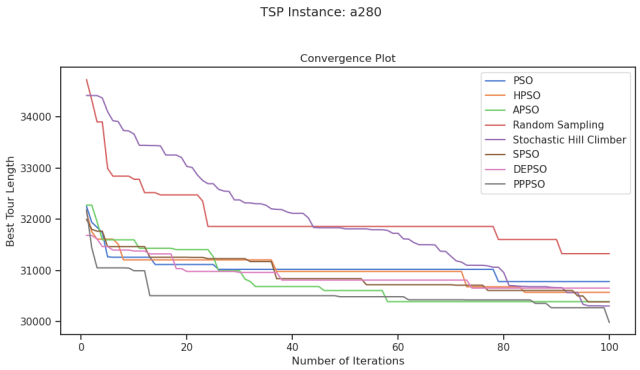

Figure 32: Hyperparmeter tuned a280

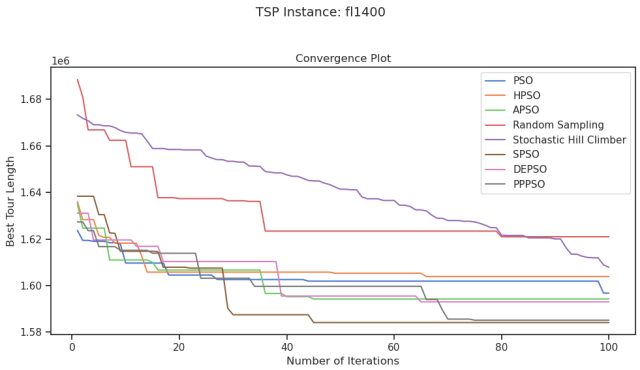

Figure 33: Hyperparameter tuned fl1400

Algorithm 4 APSO

```
Initialize particles with random positions and velocities
while termination criteria not met do
    for each particle do
        Calculate fitness value
        if fitness value is better than the best fitness value (pBest) in history then
            Set current value as the new pBest
        end if
    end for
    Choose the particle with the best fitness value of all as gBest
    Update inertia weight adaptively
    for each particle do
        Calculate particle velocity
        Update particle position
    end for
end while
```

Algorithm 5 Random Sampling

```
Initialize best solution and best fitness
for i = 1 to max_iterations do    Generatearandomsolution
    for C doalculate fitness of the random solution
        if fitness is better than best fitness then
            Update best solution and best fitness
        end if
    end for
    return best solution and best fitness
```

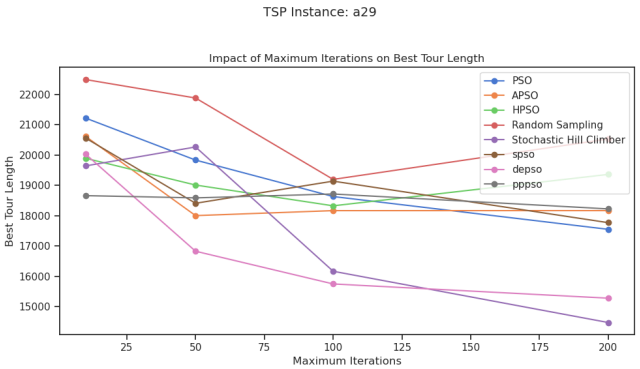

Figure 28: Max Iterations a29

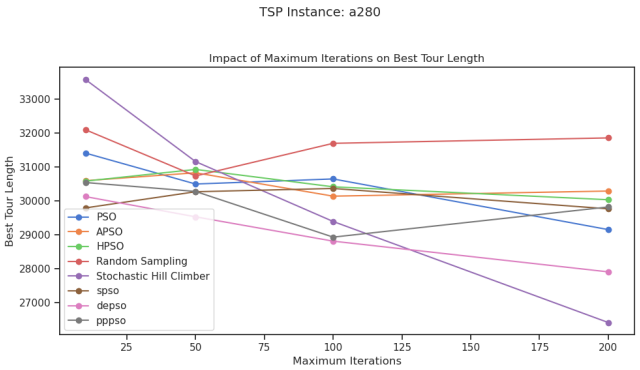

Figure 29: Max Iterations a280

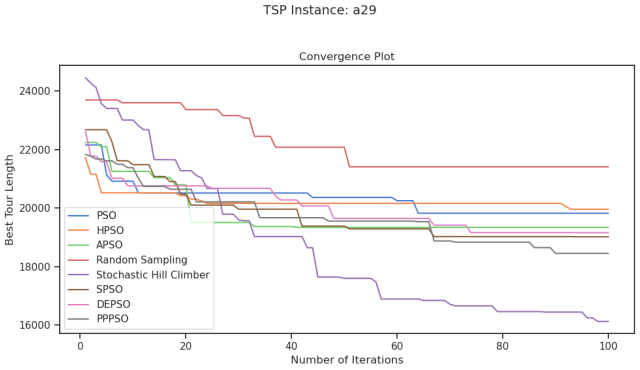

Figure 31: Hyperparameter tuned a29

Algorithm 6 Stochastic Hill Climber

```
Initialize current solution randomly
Initialize best solution and best fitness
while termination criteria not met do
    Generate a neighboring solution
    Calculate fitness of the neighboring solution
    if fitness is better than current fitness then
        Update current solution and current fitness
        if current fitness is better than best fitness then
            Update best solution and best fitness
        end if
    end if
end while
return best solution and best fitness =0
```

Algorithm 7 DEPSO

```
Initialize particles with random positions and velocities
while termination criteria not met do
    for each particle do
        Select three random particles (r1, r2, r3)
        Generate a mutant vector using DE/rand/1 strategy
        Perform crossover between the current particle and mutant vector
        Calculate fitness of the trial vector
        if trial vector fitness is better than current particle fitness then
            Replace current particle with the trial vector
        end if
        if trial vector fitness is better than pBest fitness then
            Update pBest position and fitness
        end if
    end for
    Update gBest position and fitness
end while
```

Algorithm 8 PPPSO

```
Initialize particles with random positions and velocities
Initialize predator position randomly
while termination criteria not met do
    for each particle do
        Calculate fitness value
        if fitness value is better than the best fitness value (pBest) in history then
            Set current value as the new pBest
        end if
    end for
    Choose the particle with the best fitness value of all as gBest
    for each particle do
        Calculate particle velocity considering predator influence
        Update particle position
    end for
    Update predator position based on gBest
end while
```

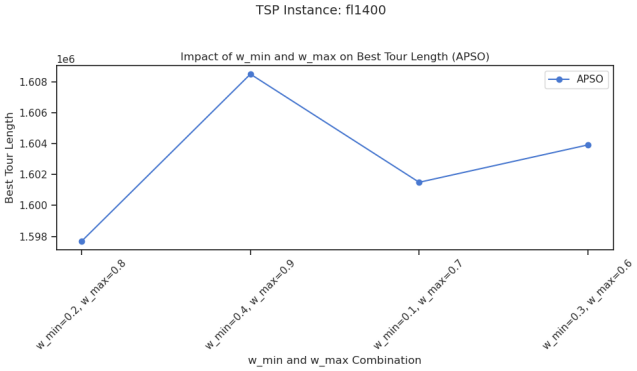

Figure 18: $Wmin$,$Wmax$ **fl1400**

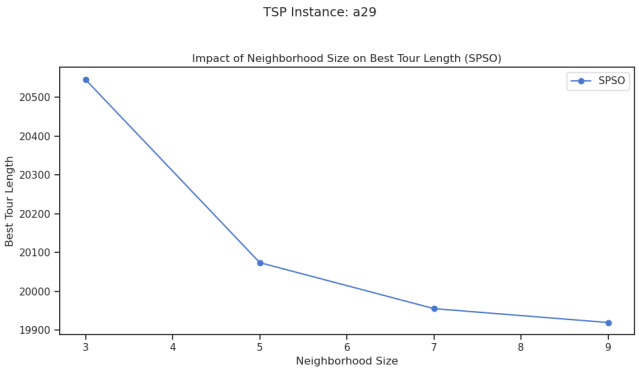

Figure 19: Neighbourhood Size a29

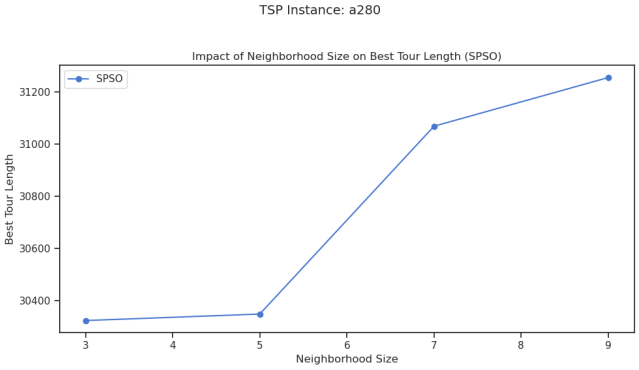

Figure 20: Neighbourhood Size a280

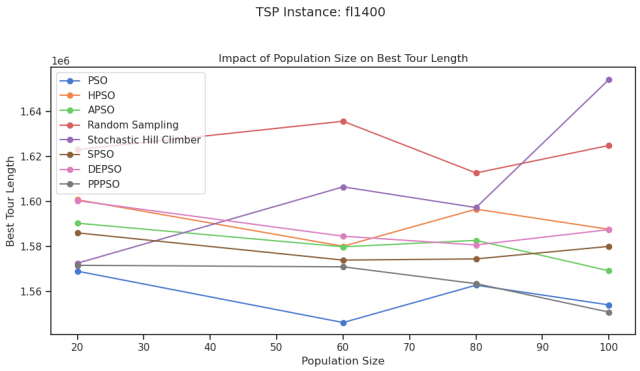

Figure 8: Population Size fl1400

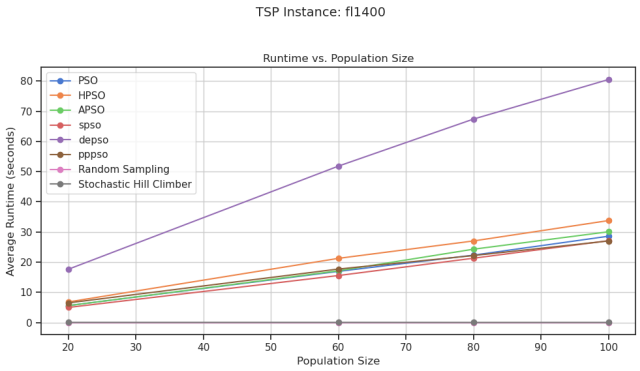

Figure 9: Runtime fl1400

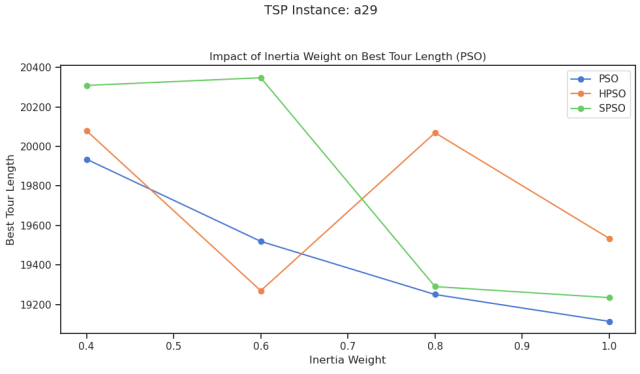

Figure 10: Inertia Weight a29

Algorithm 1 PSO

Initialize particles with random positions and velocities
while termination criteria not met **do**
  for each particle **do**
    Calculate fitness value
    if fitness value is better than the best fitness value (*pBest*) in history **then**
      Set current value as the new *pBest*
    end if
  end for
  Choose the particle with the best fitness value of all as *gBest*
  for each particle **do**
    Calculate particle velocity
    Update particle position
  end for
end while

Algorithm 2 HPSO

Initialize particles with random positions and velocities
while termination criteria not met **do**
  for each particle **do**
    Calculate fitness value
    if fitness value is better than the best fitness value (*pBest*) in history **then**
      Set current value as the new *pBest*
    end if
  end for
  Choose the particle with the best fitness value of all as *gBest*
  for each particle **do**
    Calculate particle velocity
    Update particle position using hybrid approach
  end for
end while

Algorithm 3 SPSO

Initialize particles with random positions and velocities
while termination criteria not met **do**
  for each particle **do**
    Calculate fitness value
    if fitness value is better than the best fitness value (*pBest*) in history **then**
      Set current value as the new *pBest*
    end if
    Find neighboring particles
    Update *lBest* based on neighboring particles
  end for
  for each particle **do**
    Calculate particle velocity using *lBest*
    Update particle position
  end for
end while