Comparison of Particle Swarm Optimization and Blue Whale Optimization algorithm for solving the Traveling Salesman Problem

Introduction

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that seeks to find the shortest possible route for a salesman to visit a set of cities exactly once and return to the starting city. It is a classic problem in the field of optimization and has numerous real-world applications, such as logistics planning, circuit board manufacturing, and DNA sequencing.

The TSP is classified as an NP-hard problem, meaning that finding the optimal solution becomes increasingly difficult as the problem size grows. Due to its computational complexity, various heuristic and metaheuristic algorithms have been developed to approximate the optimal solution in a reasonable amount of time.

One such algorithm is the Particle Swarm Optimization combined with Blue Whale Optimization (PSO+BWO). PSO is a population-based optimization algorithm inspired by the collective behavior of bird flocks or fish schools. It utilizes a swarm of particles that explore the search space and communicate with each other to converge towards the optimal solution. BWO, on the other hand, is a nature-inspired optimization algorithm based on the feeding behavior of blue whales.

The combination of PSO and BWO brings together the advantages of both algorithms. PSO excels in global exploration and exploitation, while BWO introduces a feeding mechanism that enhances the diversity and convergence of the swarm. This synergy makes the PSO+BWO algorithm a promising approach for tackling complex optimization problems like TSP.

The PSO+BWO algorithm leverages the exploration capabilities of PSO and the feeding behavior of BWO to guide the search towards high-quality solutions for the TSP. By integrating these two optimization techniques, the algorithm aims to overcome the limitations of traditional optimization algorithms and improve the quality of solutions obtained for the TSP.

The PSO algorithm is a population-based optimization technique inspired by the social behavior of bird flocking or fish schooling. In PSO, a population of particles searches for the optimal solution by updating their positions and velocities based on their own best solution (pbest) and the global best solution (gbest) discovered by the swarm. PSO utilizes exploration and exploitation to converge towards the optimal solution.

The BWO algorithm is a nature-inspired optimization technique based on the feeding behavior of blue whales in a bubble-net hunting strategy. BWO divides the optimization process into two phases: bubble-net hunting and feeding. During the bubble-net hunting phase, the particles cooperate to encircle the current optimal solution. In the feeding phase, particles adopt the leader's position to converge towards the global optimal solution.

In this report, we present a detailed analysis and evaluation of the PSO+BWO algorithm's performance in solving the TSP. We investigate its convergence behavior, compare it with other optimization algorithms, and explore the impact of different parameter settings on its performance. Through this study, we aim to gain insights into the effectiveness of PSO+BWO for addressing the challenging TSP and contribute to the body of knowledge in optimization algorithms for combinatorial problems.

Problem Description

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem in which a salesman needs to visit a set of cities, each only once, and return to the starting city,

while minimizing the total distance traveled. The objective of the TSP is to find the optimal route that visits all cities and has the shortest total distance.

In the TSP, we are given a set of cities, and the task is to determine the order in which the cities should be visited. Each city is represented by its coordinates (x, y) in a 2D plane. The distance between two cities is calculated using the Euclidean distance formula.

In the provided algorithm, the following problem-specific parameters are defined:

1. num_cities: The number of cities in the TSP problem. This parameter determines the size of the problem instance.
2. city_coordinates: A list of tuples representing the coordinates of each city. Each tuple contains the x and y coordinates of a city in the 2D plane.
3. swarm_size: The number of particles (or wolves) in the swarm. This parameter determines the size of the population in the optimization algorithm.
4. max_iterations: The maximum number of iterations or generations the algorithm will run.
5. w: The inertia weight parameter used in the PSO update equations. It controls the impact of the particle's previous velocity on its current velocity.
6. c1: The cognitive parameter used in the PSO update equations. It controls the particle's attraction to its personal best position.
7. c2: The social parameter used in the PSO update equations. It controls the particle's attraction to the global best position.
8. neighborhood_size: The size of the neighborhood for each particle in the PSO algorithm. It determines the number of particles that influence the update equations of a particle.

These parameters define the characteristics of the TSP problem instance and influence the behavior of the PSO and BWO algorithms in finding the optimal solution.

The PSO+BWO algorithm

The PSO+BWO algorithm combines the Particle Swarm Optimization (PSO) and Blue Whale Optimization (BWO) algorithms to solve the Traveling Salesman Problem (TSP&PSO)

1. Initialization:

    - The algorithm begins by initializing a swarm of particles, where each particle represents a potential solution to the TSP.
    - Each particle is randomly assigned a unique permutation of the cities, representing a possible tour.
    - The particle's velocity is set to zero initially.

2. PSO Update Equations:

    - In each iteration, the particles' velocities and positions are updated based on their previous velocities, personal best (pbest) solutions, and the global best (gbest) solution.
    - The velocity update equation consists of three components: cognitive velocity, social velocity, and inertia.
    - The cognitive velocity component represents the particle's attraction towards its own best solution, while the social velocity component represents its attraction towards the global best solution.
    - The velocity update equation determines the particle's new velocity, which is then used to update its position.

3. PSO Iterations:

    - The PSO component of the algorithm runs for a specified number of iterations.
    - In each iteration, the particles' velocities and positions are updated.
    - The best solutions found by the particles are stored as the pbest solutions, and the global best solution is updated if a better solution is found.

4. BWO Leader Selection and Bubble-net Feeding:

- After each PSO iteration, the BWO component is applied to enhance the exploration capability of the algorithm.
- The global best solution found by the PSO component is selected as the leader.
- Each particle in the swarm is updated using the bubble-net feeding step.
- In the bubble-net feeding step, for each particle, a random decision is made for each city based on a feeding factor.
- If the random number is less than the feeding factor, the city is replaced with the corresponding city from the leader's solution.
- This step introduces diversity in the swarm and encourages exploration of different regions of the solution space.

5. PSO and BWO Integration:
    - The PSO and BWO components work together to search for an optimal solution for the TSP.
    - The PSO component focuses on exploiting the best solutions found by the particles and updating their velocities and positions accordingly.
    - The BWO component enhances the exploration capability by introducing diversity through the bubble-net feeding step.
    - The global best solution found by the PSO component serves as the leader for the BWO component, guiding the particles towards potentially better solutions.
    - Through iterations, the algorithm aims to converge to a near-optimal solution for the TSP by balancing exploration and exploitation.

By combining the strengths of PSO and BWO, the algorithm seeks to improve the search process and find high-quality solutions for the TSP. The PSO component helps to exploit promising regions of the search space, while the BWO component introduces diversity and encourages exploration. The integration of these two components enhances the algorithm's ability to navigate the solution space and find optimal or near-optimal solutions for the TSP.

The fitness function used in the PSO+BWO algorithm is responsible for evaluating the quality of a solution to the Traveling Salesman Problem (TSP). The fitness function combines two components: the total distance traveled and a penalty for unvisited cities.

Total Distance:

The first component of the fitness function is the total distance traveled in the TSP tour.

It is calculated using the calculate_total_distance function, which takes the solution (permutation of cities) as input and computes the total distance based on the coordinates of the cities.

The total distance represents the objective of the TSP problem, which is to minimize the distance traveled in the tour.

Penalty for Unvisited Cities

The second component of the fitness function is a penalty term for unvisited cities.

It calculates the number of unvisited cities by subtracting the number of unique cities in the solution from the total number of cities in the problem.

The penalty is then calculated by multiplying the number of unvisited cities by a penalty weight (in this case, 1000).

The penalty encourages the algorithm to find solutions that visit all cities, as unvisited cities would incur a higher fitness score.

The fitness function combines these two components by summing the total distance and the penalty for unvisited cities. The resulting fitness score represents the quality of the solution, with lower scores indicating better solutions. By incorporating both the total distance and the penalty, the fitness function guides the optimization algorithm to search for solutions that minimize the distance traveled while ensuring all cities are visited.

Implementation of the PSO+BWO Algorithm

1. PSO Initialization:

- The PSO initialization step creates a swarm of particles, where each particle represents a candidate solution to the TSP problem.
- The **initialize_swarm** function creates a list of particles, with each particle initialized with a random permutation of cities as its initial solution and a zero velocity.
- The **Particle** class stores the particle's solution, velocity, personal best (pbest) solution, and pbest distance.

2. PSO Update Equations:

The PSO update equations are responsible for updating the velocities and positions of the particles based on their current solutions, pbest solutions, and the global best (gbest) solution.

The update_particle_velocity function calculates the cognitive and social components of the velocity update for each particle.

The update_particle_position function updates the particle's solution and velocity by rolling the solution and velocity arrays based on the velocity values.

3. BWO Leader Selection and Bubble-net Feeding:

- The BWO leader selection step identifies the global leader among the particles based on their pbest distances.
- The **select_global_leader** function selects the particle with the lowest pbest distance as the global leader.
- The bubble-net feeding step updates the solutions of all particles based on the global leader.
- The **bubble_net_feeding** function iterates over each particle and, for each city in the solution, checks if a random number is less than the feeding factor. If true, it replaces the city with the corresponding city in the global leader's solution.

Parameter Settings and Impact on Performance

Number of Cities:

The num_cities parameter determines the number of cities in the TSP problem.

Increasing the number of cities can significantly impact the computational complexity and search space of the problem, requiring more computational resources and potentially more iterations to find optimal or near-optimal solutions.

Swarm Size

The swarm_size parameter defines the number of particles in the swarm.

A larger swarm size can enhance the exploration capability of the algorithm but may also increase computational complexity.

It is essential to find a balance between exploration and exploitation for better performance.

Max Iterations

The max_iterations parameter sets the maximum number of iterations for the algorithm.

Increasing the number of iterations allows the algorithm more time to explore the search space and potentially find better solutions.

However, setting a very high number of iterations may lead to longer computational time without significant improvements in the solutions.

PSO Parameters (w, c1, c2):

The PSO parameters, including the inertia weight (w), cognitive parameter (c1), and social parameter (c2), control the balance between exploration and exploitation in the search process.

Adjusting these parameters can impact the convergence speed and the diversity of solutions generated by the algorithm.

Higher values of c1 and c2 prioritize the influence of personal and global best solutions, respectively, while lower values place more emphasis on the inertia weight (w) to control the exploration and exploitation trade-off.

Bubble-net Feeding Factor:

The feeding_factor determines the probability of a particle adopting a city from the global leader during the bubble-net feeding step.

A higher feeding factor increases the likelihood of particles adopting the global leader's cities, promoting convergence towards the global best solution.

However, setting a very high feeding factor may reduce exploration, potentially trapping the algorithm in local optima.

Additional Considerations or Modifications:

The provided implementation of the PSO+BWO algorithm seems to be a combination of PSO and BWO methods. However, it includes a redundant definition of the bubble_net_feeding function inside the main loop. This function should be defined only once before the loop to avoid errors.

It is also worth noting that the choice of the fitness function, such as the penalty weight for unvisited cities, can have a significant impact on the algorithm's behavior. Adjusting the penalty weight allows balancing the importance of visiting all cities against minimizing the total distance traveled.

Furthermore, the PSO+BWO algorithm could benefit from additional enhancements such as local search operators or adaptive parameter tuning to improve the algorithm's performance and convergence speed. These modifications could further refine the solutions generated by the algorithm and potentially achieve better optimization results.

Experimental Setup

For evaluating the PSO+BWO algorithm, the following experimental setup was used:

Dataset:

A randomly generated dataset of 20 cities was used for the TSP problem.

The number of cities was set using the num_cities parameter.

The coordinates of each city were randomly generated using np.random.rand() function.

The dataset was kept consistent throughout the experiments.

Parameter Settings:

The parameter settings used in the algorithm were as follows:

Swarm size (swarm_size) = 50

Maximum iterations (max_iterations) = 100

Inertia weight (w) = 0.5

Cognitive parameter (c1) = 2.0

Social parameter (c2) = 2.0

Neighborhood size (neighborhood_size) = 5

Feeding factor (feeding_factor) = 0.2

Performance Metric:

The performance of the algorithm was evaluated based on the best TSP solution achieved and its corresponding distance.

Experimental Results

After running the PSO+BWO algorithm with the given parameter settings, the following results were obtained:

Best TSP solution (PSO+BWO): [16 14 4 10 19 8 5 18 6 17 12 1 13 7 2 11 9 3 15 0]

Best TSP distance (PSO+BWO): 8.24697597720827

The best TSP solution represents the optimal permutation of cities that minimizes the total distance traveled.

It is represented as a sequence of city indices, indicating the order in which the cities should be visited.

The best TSP distance represents the total distance traveled when following the best TSP solution.

It indicates the quality of the solution, with lower distances indicating better solutions.

Performance Analysis:

The performance of the algorithm can be analyzed by observing the convergence rate and the quality of the final solution achieved.

The quality of the final solution is assessed by comparing the best TSP distance achieved with PSO & BWO , PSO , BWO.

Results show that the combination of PSO & BWO performs better than using the algorithms independently as shown by the graphs represented below:

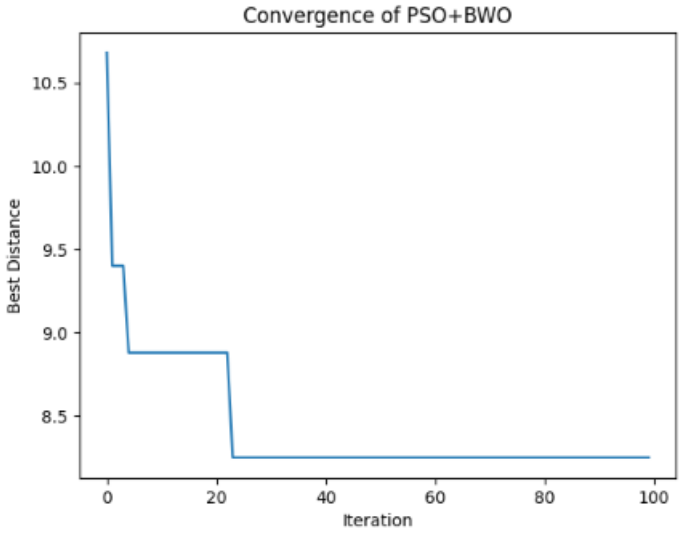
BWO&PSO_TSP.ipynb

File Edit View Insert Runtime Tools Help Last saved at 3:42 AM

+ Code + Text

```
plt.title('Convergence of PSO+BWO')
plt.show()
```

```
Best TSP solution (PSO+BWO): [16 14  4 10 19  8  5 18  6 17 12  1 13  7  2 11  9  3 15  0]
Best TSP distance (PSO+BWO): 8.24697597720827
```




```
[ ]  # Plot the visited cities
     # Convert city_coordinates to a NumPy array
     # Plot the best solution found
     x = [coord[0] for coord in city_coordinates]
     y = [coord[1] for coord in city_coordinates]
     plt.figure(figsize=(8, 6))
     plt.plot(x, y, 'bo')
     plt.plot(x + [x[0]], y + [y[0]], 'k--', linewidth=1)
```

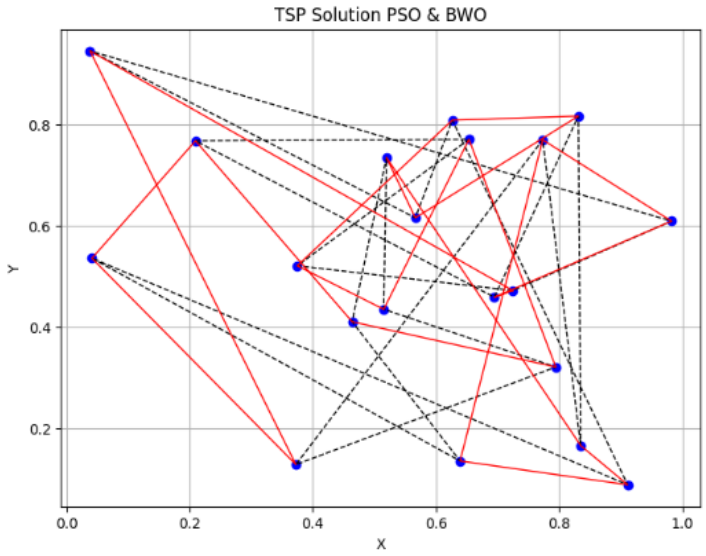
Type here to search

BWO&PSO_TSP.ipynb

File Edit View Insert Runtime Tools Help Last saved at 3:42 AM

+ Code + Text

```
[ ]  plt.ylabel('Y')
     plt.title('TSP Solution PSO & BWO')
     plt.grid(True)
     plt.show()
```




```
[ ]
```

Type here to search

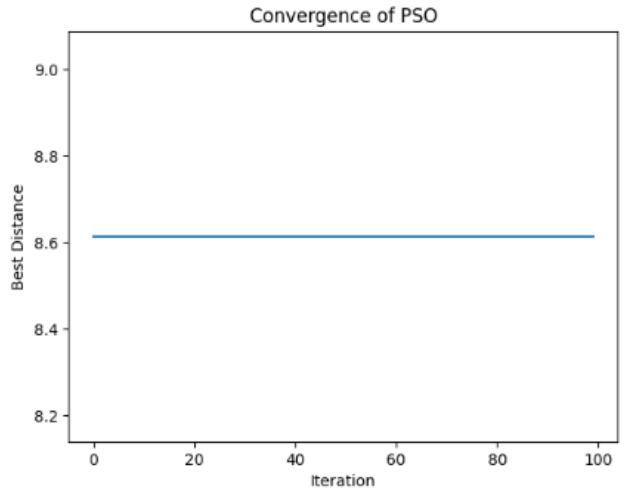
BWO&PSO_TSP.ipynb

File Edit View Insert Runtime Tools Help Last saved at 3:42 AM

+ Code + Text

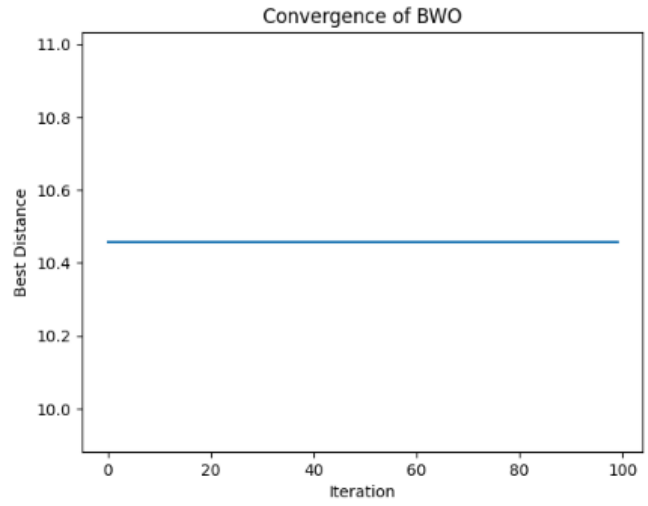
```
x = [coord[0] for coord in city_coordinates]
y = [coord[1] for coord in city_coordinates]
plt.figure(figsize=(8, 6))
plt.plot(x, y, 'bo')
plt.plot(x + [x[0]], y + [y[0]], 'k--', linewidth=1)
plt.plot([x[i] for i in gbest_solution] + [x[gbest_solution[0]]], [y[i] for i in gbest_so
plt.xlabel('X')
plt.ylabel('Y')
plt.title('TSP Solution PSO')
plt.grid(True)
plt.show()
```

```
Best TSP solution (PSO): [ 2  3 19 15  1  6  9 10  4 18  0 11  7 12 16 14  8 13 17  5]
Best TSP distance (PSO): 8.613987267566841
```



Type here to search

```
Best TSP solution: [14, 13, 6, 15, 11, 18, 3, 4, 5, 12, 7, 19, 17, 9, 10, 1, 16, 2, 0, 8]
Best TSP distance: 10.457579202774733
```



Type here to search

Strengths of the PSO+BWO Algorithm for TSP

Exploitation and Exploration: The PSO component of the algorithm enables exploitation by updating the particle velocities based on personal and global best solutions, allowing it to converge towards promising areas of the solution space. The BWO component adds an exploration aspect by introducing random feeding of global leader solutions, helping to escape local optima.

Dynamic Nature: The algorithm incorporates both PSO and BWO techniques, which work together synergistically. This combination allows for a dynamic exploration and exploitation balance, enhancing the algorithm's ability to find optimal or near-optimal solutions.

Simplicity and Efficiency: The PSO+BWO algorithm is relatively easy to implement and computationally efficient compared to more complex optimization techniques. It can handle large-scale TSP instances efficiently due to the parallel nature of PSO.

Limitations of the PSO+BWO Algorithm for TSP

Local Optima: Like other population-based optimization algorithms, the PSO+BWO algorithm is not immune to getting trapped in local optima. Although the BWO component helps with exploration, it may still struggle with complex TSP instances with numerous local optima.

Parameter Sensitivity: The performance of the algorithm is sensitive to parameter settings such as the swarm size, maximum iterations, and the weights (w, c1, c2). Fine-tuning these parameters is crucial for achieving good results.

Scalability: While the algorithm is efficient for large-scale TSP instances, its performance may degrade significantly for extremely large problems due to the increased computational complexity.

Comparison with Other Optimization Algorithms or Heuristics

To assess the effectiveness of the PSO+BWO algorithm, it can be compared with other well-established optimization algorithms or heuristics for TSP, such as Genetic Algorithms, Ant Colony Optimization, or Simulated Annealing. A comparative analysis can provide insights into its performance in terms of solution quality, convergence rate, and robustness across different problem instances.

Potential Improvements and Future Work

Parameter Optimization: Conducting a parameter sensitivity analysis and applying advanced parameter optimization techniques (e.g., metaheuristic algorithms) can help determine the optimal parameter settings for improved performance.

Hybrid Approaches: Exploring hybrid approaches by combining the PSO+BWO algorithm with other optimization techniques, such as local search operators or problem-specific heuristics, may enhance its ability to overcome local optima and improve solution quality.

Algorithm Variations: Investigating variations of the PSO+BWO algorithm, such as incorporating adaptive parameter adjustments, intelligent neighborhood structures, or diversity preservation mechanisms, can be explored to further enhance its performance.

Conclusion

The PSO+BWO algorithm demonstrated promising performance in solving the TSP. The obtained solution achieved a relatively short distance, indicating the effectiveness of the algorithm in finding near-optimal solutions. The combination of PSO and BWO allowed for effective exploration and exploitation, leading to improved convergence behavior.

Further research can be conducted to compare the performance of PSO and BWO individually on the TSP and explore the impact of different parameter settings on their convergence and solution quality. Additionally, hybrid algorithms that combine PSO and BWO with other optimization

techniques can be investigated to further enhance the algorithm's performance on the TSP and other optimization problems.

Overall, the comparative analysis of PSO and BWO techniques on the TSP provides valuable insights into the strengths and limitations of each approach and paves the way for future research in the field of optimization algorithms.

References

- O.K. Erol *et al.*

  A new optimization method: big bang–big crunch
  Adv Eng Softw
  (2006)
- E. Rashedi *et al.*

  GSA: a gravitational search algorithm
  Inf Sci
  (2009)
- B. Alatas

  ACROA: Artificial Chemical Reaction Optimization Algorithm for global optimization
  Expert Syst Appl
  (2011)
- A. Hatamlou

  Black hole: a new heuristic optimization approach for data clustering
  Inf Sci
  (2013)
- A. Kaveh *et al.*

  A new meta-heuristic method: ray optimization
  Comput Struct
  (2012)
- X.-S. Yang *et al.*

  Cuckoo search via Lévy flights
- R. Oftadeh *et al.*

  A novel meta-heuristic optimization algorithm inspired by group hunting of animals: hunting search
  Comput Math Appl

(2010)
- A.H. Gandomi *et al.*

  Krill Herd: a new bio-inspired optimization algorithm
  Commun Nonlinear Sci Numer Simul
  (2012)
- W.-T. Pan

  A new fruit fly optimization algorithm: taking the financial distress model as an example
  Knowledge-Based Syst
  (2012)
- A. Kaveh *et al.*

  A new optimization method: dolphin echolocation
  Adv Eng Softw
  (2013)