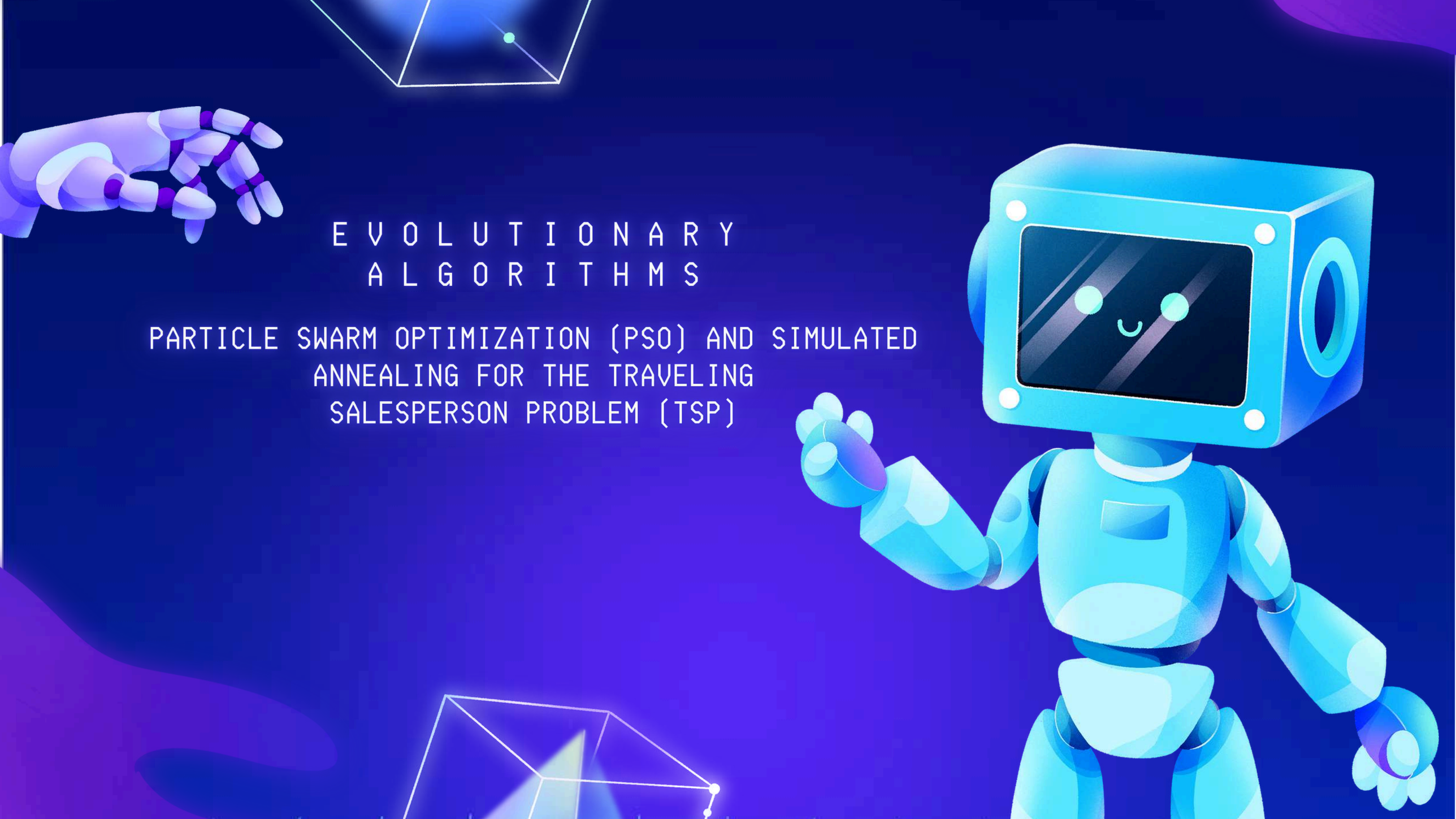
EVOLUTIONARY ALGORITHMS

PARTICLE SWARM OPTIMIZATION (PSO) AND SIMULATED ANNEALING FOR THE TRAVELING SALESPERSON PROBLEM (TSP)



WHAT IS THE TRAVEL SALESMAN PROBLEM

The Traveling Salesman Problem (TSP) is a classic problem in combinatorial optimization and computer science

---

Given a list of cities and the distances between each pair of cities, the objective of the TSP is to find the shortest possible route that visits each city exactly once and returns to the original city.

In other words, the TSP seeks to determine the most efficient way for a traveling salesman to visit all the cities in their territory and return to their starting point, minimizing the total distance traveled

REAL-WORLD APPLICATIONS

Delivery Route Planning

Delivery companies use TSP algorithms to optimize delivery routes, minimizing travel time and fuel costs.

Circuit Board Drilling

In manufacturing printed circuit boards (PCBs), TSP helps determine the most efficient path for a drilling machine to drill holes on the board.

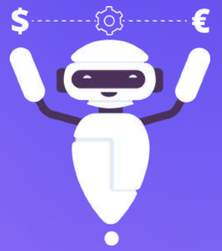
Machine Scheduling

TSP can be applied to schedule the order in which a robot arm needs to visit different points on a production line.

SOLUTION APPROACHES

METAHEURISTICS

: These are advanced search algorithms inspired by natural phenomena like simulated annealing or genetic algorithms. They can find near-optimal solutions for large problems.



HEURISTICS

These are approximate algorithms that provide good (but not guaranteed to be optimal) solutions in a reasonable amount of time. Examples include nearest neighbor, insertion heuristics, and 2-opt and 3-opt exchanges.

BRUTE-FORCE SEARCH

This is a straightforward approach where you calculate the distance of every possible route and choose the shortest one. However, it becomes impractical for large datasets due to the exponential explosion of possibilities.

SWARM INTELLIGENCE

SWARM INTELLIGENCE (SI) IS A FASCINATING FIELD OF ARTIFICIAL INTELLIGENCE (AI) THAT STUDIES THE COLLECTIVE BEHAVIOR OF DECENTRALIZED, SELF-ORGANIZED SYSTEMS IN NATURE. THESE SYSTEMS, LIKE FLOCKS OF BIRDS, SCHOOLS OF FISH, OR COLONIES OF ANTS, EXHIBIT REMARKABLE COORDINATION AND PROBLEM-SOLVING ABILITIES DESPITE HAVING RELATIVELY SIMPLE INDIVIDUAL BEHAVIORS.

BENEFITS OF SWARM INTELLIGENCE ALGORITHMS

ROBUSTNESS

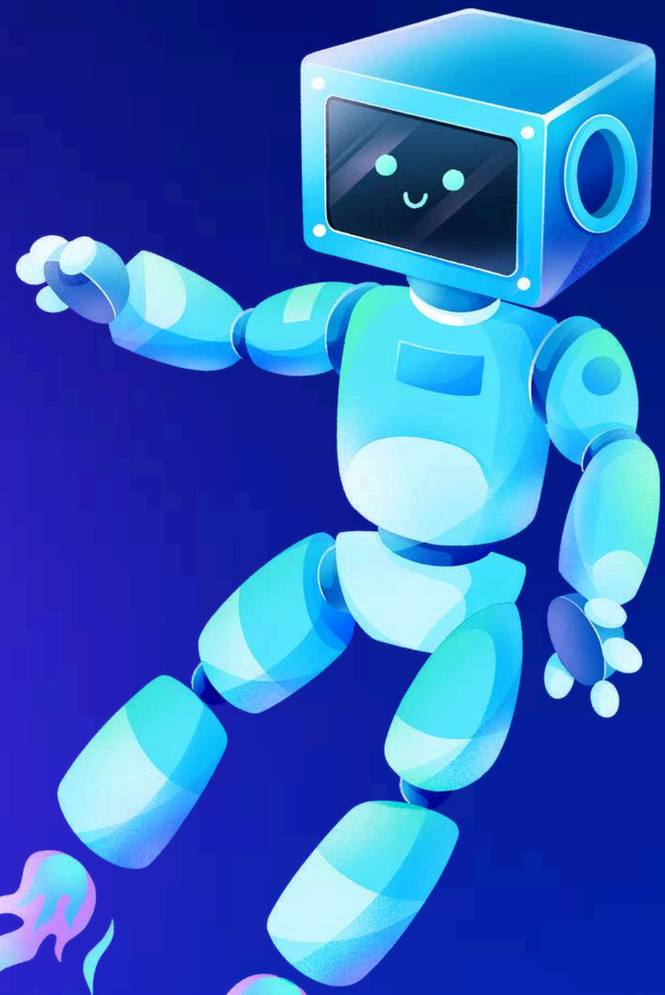
Decentralized nature makes them resilient to failures of individual agents.

SCALABILITY

Algorithms can be easily adapted to problems with large numbers of agents.

FLEXIBILITY

They can handle dynamic environments and problems with no pre-defined paths to solutions.



PARTICLE SWARM OPTIMIZATION

Is a computational technique inspired by the collective movement of swarms in nature, such as flocks of birds or schools of fish.

BENEFITS OF PSO

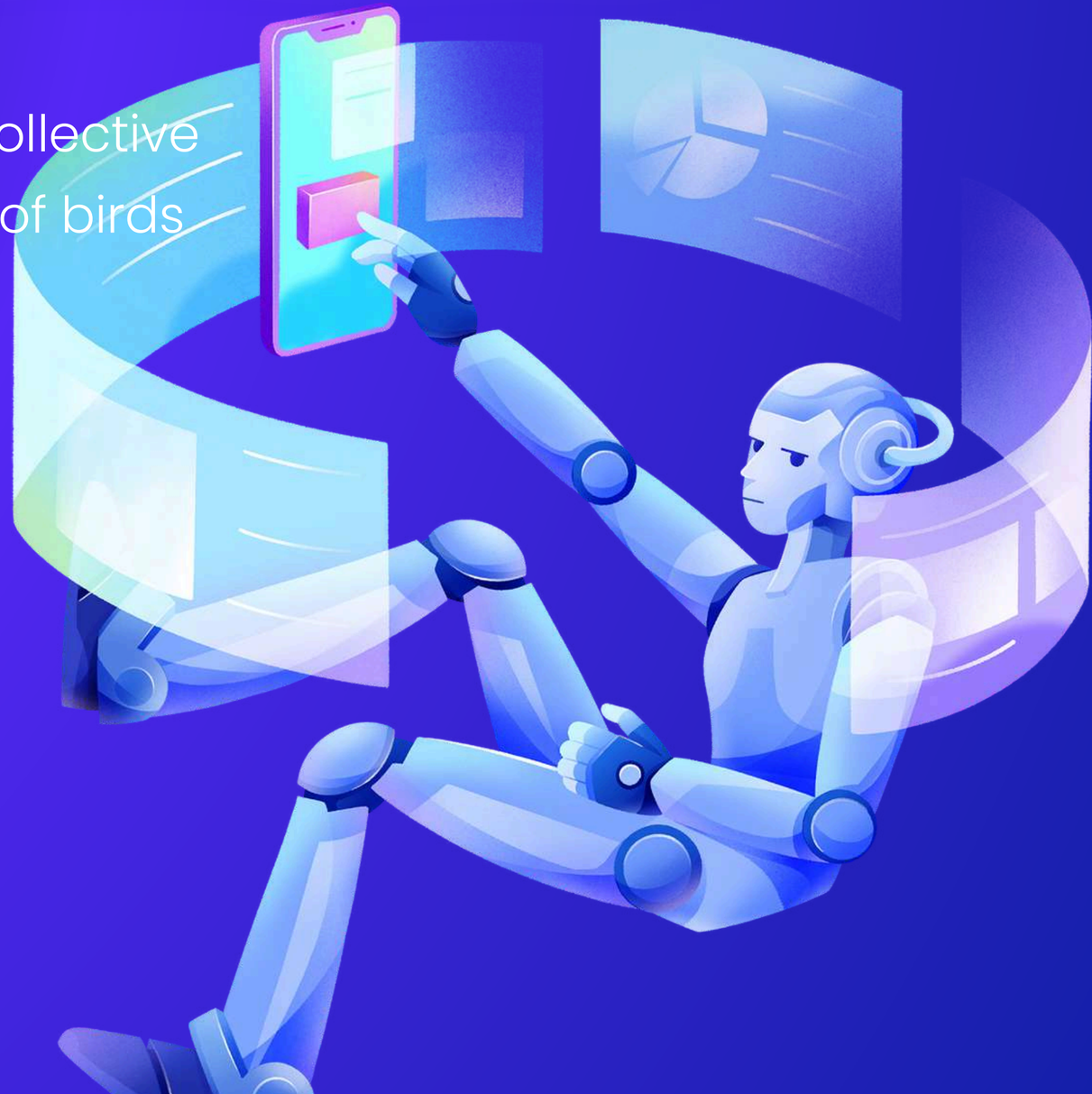
- iRelatively simple to implement compared to some other optimization algorithms
- Efficient for problems with continuous search spaces.
- Can handle problems with a large number of dimensions (parameters)

LIMITATIONS OF PSO

- May struggle to converge to the optimal solution in complex problems with many local optima (deceptive valleys in the search space).
- Fine-tuning parameters like inertia weights and learning rates can be crucial for good performance.

PARTICLE SWARM OPTIMIZATION

is a computational technique inspired by the collective movement of swarms in nature, such as flocks of birds or schools of fish.



INSPIRATION FROM NATURE:

- IMAGINE A FLOCK OF BIRDS SEARCHING FOR FOOD. EACH BIRD REPRESENTS A POTENTIAL SOLUTION IN YOUR OPTIMIZATION PROBLEM.EXPAND_MORE
- THE BIRDS DON'T HAVE A MAP OR A PRE-DEFINED PATH. THEY RELY ON THEIR OWN EXPERIENCE (FINDING GOOD FOOD SOURCES) AND COMMUNICATION WITH THEIR NEIGHBORS (SHARING INFORMATION ABOUT DISCOVERED RESOURCES).

APPLICATIONS OF PSO

01

- Engineering design optimization (e.g., finding the most efficient structure for a bridge)

02

- Machine learning (e.g., hyperparameter tuning for neural networks)

03

- Robotics control (e.g., optimizing movement patterns for robots)

04

- Financial portfolio optimization (e.g., finding the best asset allocation)

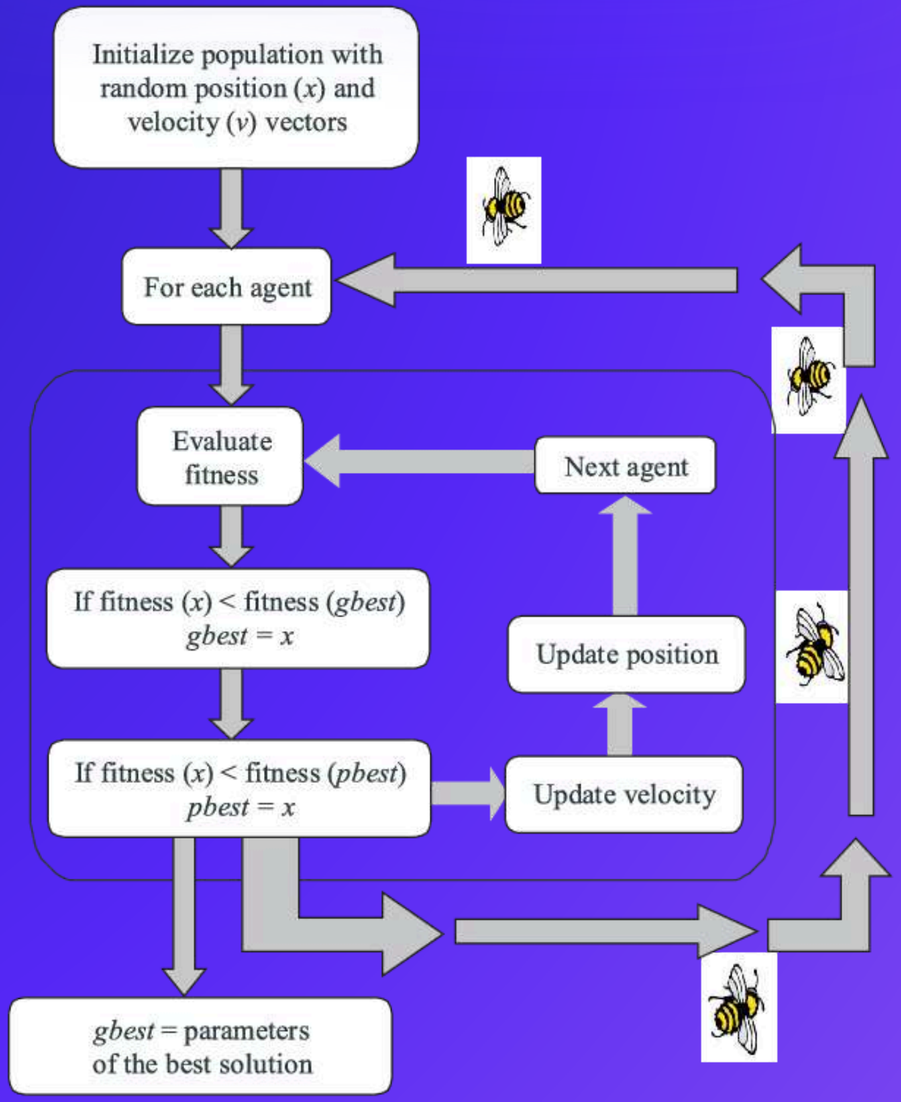
PSO ALGORITHM

1. Initialization:
   - A population of particles (potential solutions) is created. Each particle represents a candidate solution to the problem being optimized.
   - Each particle is assigned a random position in the search space, which defines the candidate solution it represents.
   - The particles are also assigned an initial velocity, which determines the direction and magnitude of their movement in the search space.
2. Fitness Evaluation:
   - Each particle's fitness is evaluated based on a predefined objective function. This function measures how good a particular solution is.
3. Personal Best Update:
   - Each particle keeps track of its own best position (personal best) encountered so far, based on its fitness evaluation.
4. Global Best Update:
   - The entire swarm also keeps track of the best position found by any particle in the population (global best).
5. Velocity Update:
   - The velocity of each particle is updated based on its current velocity, its attraction towards its personal best position, and its attraction towards the global best position. The update typically involves:Inertia: A term that encourages particles to continue moving in the same direction.
   - Cognitive Component: A term that pulls the particle towards its personal best position.
   - Social Component: A term that pulls the particle towards the global best position.
6. Movement:
   - Based on the updated velocity, each particle moves to a new position in the search space.
7. Iteration:
   - Steps 2 to 6 are repeated for a predetermined number of iterations.
8. Termination:
   - The algorithm terminates after reaching a maximum number of iterations or when a satisfactory solution is found.

PSO PARTICLES MOVEMENT

$$\begin{cases} V(t+1) = C(V(t) + c_1 rand_1 * (Pbest_i - X(t)) \\ \qquad\qquad + c_2 rand_2 * (Gbest_i - X(t)) \\ X(t+1) = X(t) + V(t+1) \end{cases}$$

PSO FLOWCHART



PSO PSEUDO CODE

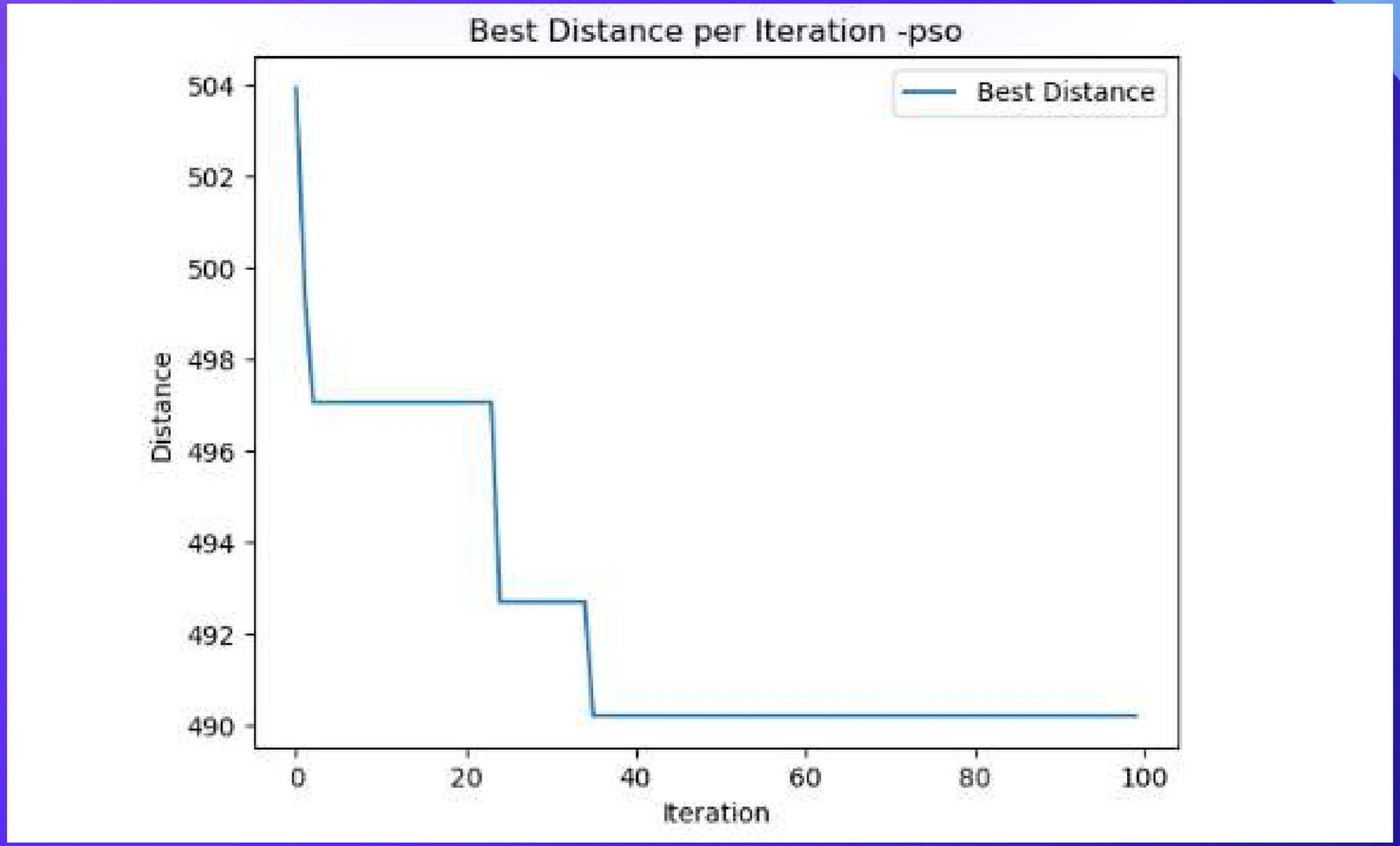
```
Procedure PSO
 Initialize particles population
do
        for each particle p with position x_p do
                calculate fitness value f(x_p)
                if  f(x_p) is better than pbest_p then
                        pbest_p ← x_p
                endif
        endfor

        Define gbest_p as the best position found so far by any of
        p's neighbors

        for each particle p do
                v_p ← compute_velocity(x_p, pbest_p, gbest_p)
                x_p ← update_ position(x_p, v_p)
        endfor

while   (Max iteration is not reached or a stop criterion is
         not satisfied)
```

PSO CONVERGENCE



SIMULATED ANNEALING

Simulated Annealing (SA) is a probabilistic optimization technique inspired by the annealing process in metallurgy. Just like how slowly cooling a metal mixture allows atoms to arrange themselves in a more stable, lower energy state, SA helps find near-optimal solutions for complex problems by allowing some "bad" moves during the search.

BENEFITS OF SA

- Escapes Local Optima: By allowing uphill moves with some probability, SA can escape from getting stuck in suboptimal solutions.
- Fewer Parameters: Compared to some other optimization algorithms, SA requires tuning only a few parameters (initial temperature and cooling schedule).

LIMITATIONS OF SA

- 1. Slow Convergence: SA can be computationally expensive due to its iterative nature and the need to explore a large portion of the search space. The cooling schedule plays a crucial role. A slow cooling schedule allows for more exploration but takes longer to converge. A fast cooling schedule might converge faster but could lead to getting stuck in local optima. Finding the right balance can be challenging.
- 2. Sensitive to Cooling Schedule: The effectiveness of SA heavily relies on the chosen cooling schedule. An overly aggressive cooling schedule (temperature reduction) can cause the algorithm to get trapped in local optima, while a slow schedule might lead to excessive computation time. Tuning the cooling schedule for optimal performance can be complex and problem-specific.

APPLICATIONS OF SA

01

- VLSI (Very-Large-Scale Integration) Circuit Design: Optimizing placement and routing of circuits.

02

- Traveling Salesman Problem (TSP): Finding the shortest possible route that visits each city exactly once.

03

- Signal Processing: Optimizing filter parameters for noise reduction or image enhancement.

04

- Finance: Portfolio optimization.

SA ALGORITHM

- 1-Initial State:
- Define the initial solution (candidate configuration) for the optimization problem.
- Set an initial temperature (T), controlling the acceptance probability of worse solutions.
- 2-Iteration Loop:
- Repeat the following steps for a certain number of iterations:
  - Generate Neighbor: Create a small random modification of the current solution, resulting in a new neighboring solution.
  - Calculate Delta E: Compute the difference in objective function values (energy difference) between the current solution and the neighboring solution (ΔE).
  - Acceptance Probability: Decide whether to accept the neighboring solution based on a probability function that depends on ΔE and the current temperature (T).
    - If ΔE is negative (improvement), the new solution is always accepted.
    - If ΔE is positive (worsening), it's still accepted with a probability based on the Boltzmann distribution, which decreases as the temperature cools and ΔE increases. This allows for exploration of the search space and escape from local optima.
- 3-Temperature Update:
- Gradually decrease the temperature (T) according to a cooling schedule. This reduces the probability of accepting worse solutions as the algorithm progresses.
- 4-Termination:
- Stop the loop when a certain stopping criterion is met, such as reaching a maximum number of iterations or finding a sufficiently good solution.

SA ACCEPTANCE FUNCTION

Algorithm 1: Acceptance Function

Data: T, ΔE - the temperature and the energy variation between the new candidate solution and the current one.

Result: Boolean value that indicates if the new solution is accepted or rejected.

if $(\Delta E < 0)$ **then**
    return True;
else
    $r \leftarrow$ generate a random value in the range [0, 1)
    if $(r < \exp(-\Delta E/T))$ **then**
        return True
    else
        return False
    end
end

SA FLOWCHART

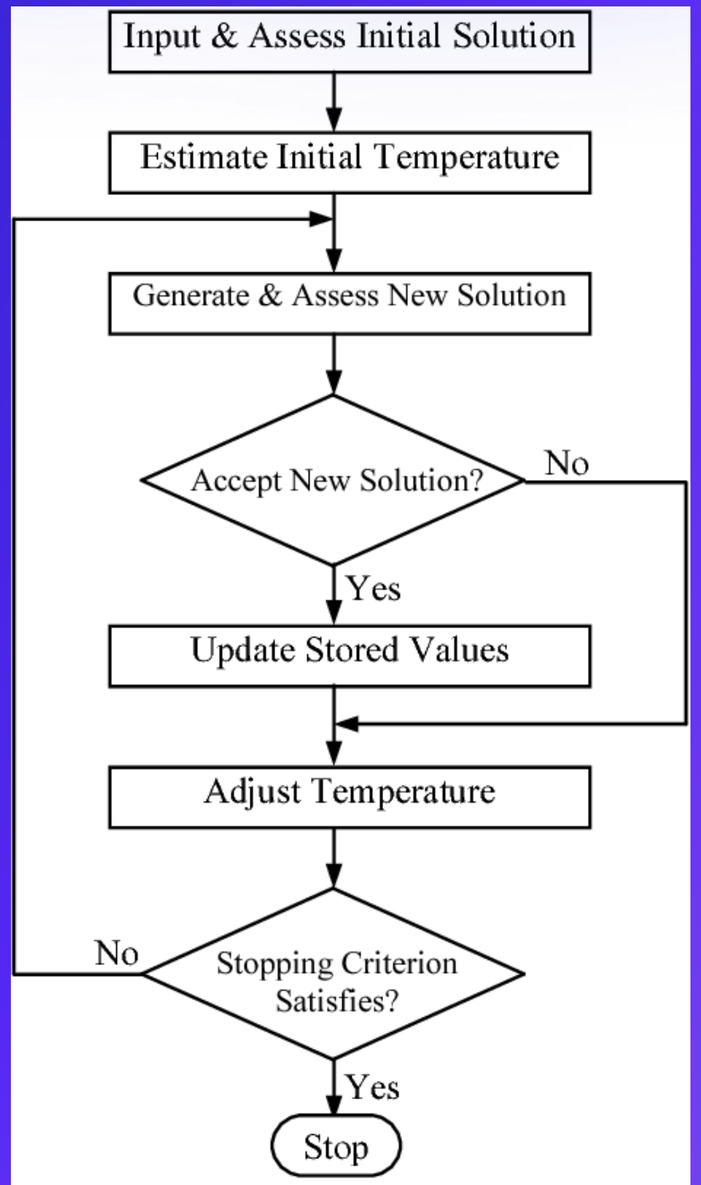


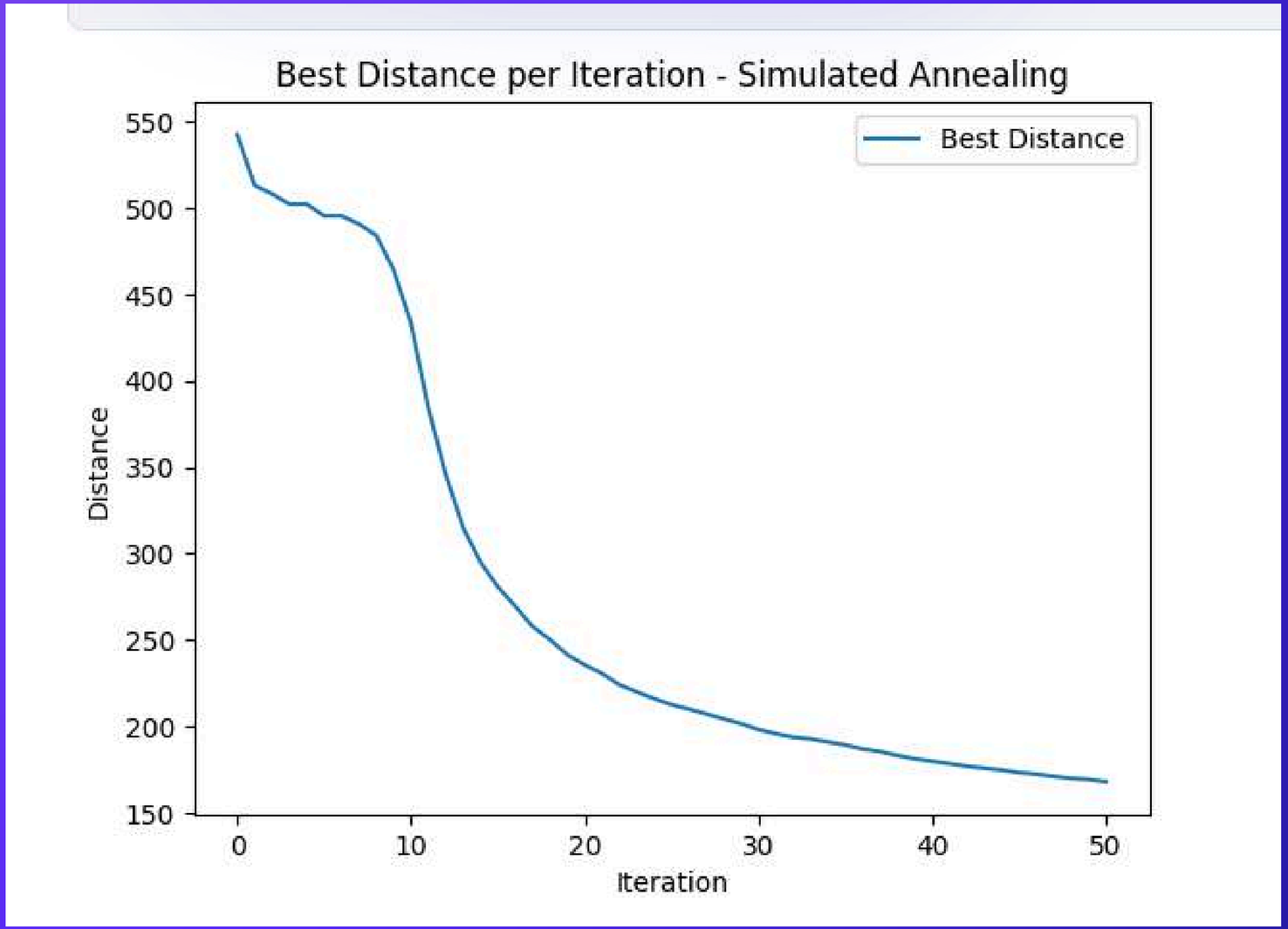
Fig. 1 The flow chart of a SA

SA PSEUDO CODE

Simulated annealing algorithm

```
1  Select the best solution vector x0 to be optimized
2  Initialize the parameters: temperature T, Boltzmann's constant k, reduction factor c
3  while termination criterion is not satisfied do
4      for number of new solution
6          Select a new solution: x0+Δx
7              if f(x0+Δx) > f(x0) then
8                  f_new = f(x0 + Δx);   x0 = x0 + Δx
9                  else
10                     Δf = f(x0 + Δx) − f(x0)
11                     random r(0, 1)
12                         if r > exp(−Δf/kT) then
13                             f_new = f(x0 + Δx),   x0 = x0 + Δx
14                             else
15                             f_new = f(x0),
16                         end if
17             end if
18         f = f_new
19         Decrease the temperature periodically: T = c x T
20     end for
21 end while
```

SA CONVERGENCE



PSO VS SA

algorithm	Inspiration	Search Strategy	Strengths	Weaknesses	Suitable for
PSO	Swarm behavior (flocks, schools)	Velocity and personal/global best info	Simple, few parameters, fast convergence	Can get stuck in local optima, parameter-sensitive	Well-defined problems, fast convergence
SA	Annealing process in metallurgy	Probability-based, allows "bad" moves	Escapes local optima, few parameters	Slower, needs careful cooling schedule	Complex problems, escaping local optima

PSO VS SA AFTER DIFFERENT 30 RUNS

1-PSO:

```
Minimum value with  w = 0.6903230537823498 , c1 = 1.6163512649298795 , c2 = 2.3836487350701203 : 488.78454688056
Minimum value with  w = 0.5199701970478606 , c1 = 1.1463777238097004 , c2 = 2.8536222761902996 : 495.74803220004
Minimum value with  w = 0.8697732799442692 , c1 = 2.535073680264377 , c2 = 1.464926319735623 : 493.5414697741501
Minimum value with  w = 0.8986318600679887 , c1 = 1.4439967259004522 , c2 = 2.556003274099548 : 495.141509546055
Minimum value with  w = 0.7043016666646792 , c1 = 1.3198356009950585 , c2 = 2.6801643990049415 : 494.04756338020
Minimum value with  w = 0.9958837123018935 , c1 = 2.464864200258197 , c2 = 1.535135799741803 : 494.0566408791760
Minimum value with  w = 0.24396261573937494 , c1 = 1.19982855785125 , c2 = 2.8001714421487502 : 494.814477167205
Minimum value with  w = 0.16941380310500131 , c1 = 1.778286755045267 , c2 = 2.221713244954733 : 494.683298146963
Minimum value with  w = 0.5241416166700923 , c1 = 2.851252828706482 , c2 = 1.1487471712935182 : 493.115602635731
Minimum value with  w = 0.34117056577632243 , c1 = 2.414286436641799 , c2 = 1.5857135633582011 : 491.12088700282
Minimum value with  w = 0.24628817083836885 , c1 = 1.5928776761830656 , c2 = 2.4071223238169344 : 494.4938272512
Minimum value with  w = 0.681848736665844 , c1 = 1.3250716678592136 , c2 = 2.6749283321407864 : 493.610039739134
Minimum value with  w = 0.19747358052159092 , c1 = 2.623117446932002 , c2 = 1.3768825530679978 : 485.70282314572
Minimum value with  w = 0.34363083971320474 , c1 = 1.1320085917307974 , c2 = 2.8679914082692024 : 495.5274292837
Minimum value with  w = 0.397082551105034 , c1 = 1.1341982556178705 , c2 = 2.8658017443821295 : 495.939698606620
Minimum value with  w = 0.006468467619046181 , c1 = 1.665117921461216 , c2 = 2.334882078538784 : 491.63455817305
Minimum value with  w = 0.09349491086511907 , c1 = 1.4933406976681218 , c2 = 2.506659302331878 : 491.78865507670
Minimum value with  w = 0.3693670068578231 , c1 = 1.9654056618434097 , c2 = 2.03459433815659 : 491.1305127021205
Minimum value with  w = 0.024613888634420045 , c1 = 2.7578270825168323 , c2 = 1.2421729174831677 : 488.394910857
Minimum value with  w = 0.2920954839523692 , c1 = 2.4224877857997855 , c2 = 1.5775122142002145 : 493.82905723417
Minimum value with  w = 0.3086190834932845 , c1 = 1.1867568908816624 , c2 = 2.8132431091183374 : 491.73441324994
Minimum value with  w = 0.05909641268763166 , c1 = 1.8249813752860224 , c2 = 2.1750186247139776 : 490.1320455282
Minimum value with  w = 0.7471286098960029 , c1 = 1.285359306622114 , c2 = 2.714640693377886 : 489.5539382760884
Minimum value with  w = 0.19422663034356513 , c1 = 1.2744422476407467 , c2 = 2.7255577523592533 : 492.4596907673
Minimum value with  w = 0.43972858584156627 , c1 = 2.582775676364397 , c2 = 1.4172243236356028 : 490.38829931840
Minimum value with  w = 0.7126634755678545 , c1 = 1.8204401119810307 , c2 = 2.1795598880189693 : 495.38197980539
Minimum value with  w = 0.8139052198850137 , c1 = 2.3228926256158475 , c2 = 1.6771073743841525 : 486.62804245681
Minimum value with  w = 0.5535353994168714 , c1 = 2.3086728104673195 , c2 = 1.6913271895326805 : 493.25732296381
Minimum value with  w = 0.49201350375364516 , c1 = 2.421628853975527 , c2 = 1.5783711460244731 : 489.81484773975
Minimum value with  w = 0.055709682213928136 , c1 = 2.1913785599248783 , c2 = 1.8086214400751217 : 494.407264032
```

2-SA:

```
Minimum value with temperature= 8 , cooling_rate = 0.22924597564673832 , i = 50  : 165.18746083751168
Minimum value with temperature= 44 , cooling_rate = 0.46131856870561283 , i = 50  : 157.20155654892295
Minimum value with temperature= 35 , cooling_rate = 0.8913392217373209 , i = 50  : 161.66883302557085
Minimum value with temperature= 67 , cooling_rate = 0.65112920805402 , i = 50  : 163.41869750377015
Minimum value with temperature= 97 , cooling_rate = 0.6401655628l6801 , i = 50  : 162.97172061445806
Minimum value with temperature= 39 , cooling_rate = 0.4233826767520904 , i = 50  : 160.50877384321177
Minimum value with temperature= 62 , cooling_rate = 0.206721049503509 , i = 50  : 160.36035448065311
Minimum value with temperature= 79 , cooling_rate = 0.18091043512549898 , i = 50  : 163.24268775059394
Minimum value with temperature= 60 , cooling_rate = 0.07088704021351677 , i = 50  : 160.5869417991241
Minimum value with temperature= 23 , cooling_rate = 0.0030785576274521276 , i = 50  : 165.92374550413655
Minimum value with temperature= 45 , cooling_rate = 0.6282541084431252 , i = 50  : 155.55871469380693
Minimum value with temperature= 79 , cooling_rate = 0.6013665623957405 , i = 50  : 160.3569739189919
Minimum value with temperature= 20 , cooling_rate = 0.6437407991542364 , i = 50  : 165.36338347918735
Minimum value with temperature= 26 , cooling_rate = 0.4397862445809311 , i = 50  : 163.56932150909321
Minimum value with temperature= 10 , cooling_rate = 0.15858145993361528 , i = 50  : 163.22487076346226
Minimum value with temperature= 29 , cooling_rate = 0.15059901684104526 , i = 50  : 167.16667876277933
Minimum value with temperature= 42 , cooling_rate = 0.6123752646454338 , i = 50  : 154.8320912116949
Minimum value with temperature= 46 , cooling_rate = 0.48689605034694483 , i = 50  : 162.51636543254529
Minimum value with temperature= 51 , cooling_rate = 0.10847405310673619 , i = 50  : 163.39083695193335
Minimum value with temperature= 47 , cooling_rate = 0.22985714774761734 , i = 50  : 162.71084259579672
Minimum value with temperature= 45 , cooling_rate = 0.5250993894164839 , i = 50  : 159.253611194504
Minimum value with temperature= 53 , cooling_rate = 0.847328678803967 , i = 50  : 160.6072669863501
Minimum value with temperature= 59 , cooling_rate = 0.4662138329111609 , i = 50  : 167.0365612304077
Minimum value with temperature= 85 , cooling_rate = 0.44467514686350074 , i = 50  : 166.23304308019684
Minimum value with temperature= 64 , cooling_rate = 0.5973746609731911 , i = 50  : 164.43686130911667
Minimum value with temperature= 38 , cooling_rate = 0.5854984383120161 , i = 50  : 161.2497328690773
Minimum value with temperature= 82 , cooling_rate = 0.16566400900280243 , i = 50  : 159.76078648424954
Minimum value with temperature= 92 , cooling_rate = 0.33225015222375487 , i = 50  : 164.1724346095067
Minimum value with temperature= 15 , cooling_rate = 0.09028559615967235 , i = 50  : 157.9763110936145
Minimum value with temperature= 8 , cooling_rate = 0.43314788723600806 , i = 50  : 161.01623642865246
```

OTHER ALGORITHMS USED

1-GENETICS ALGORITHM

Genetic algorithms (GA) are a type of evolutionary algorithm inspired by the process of natural selection. They are well-suited for solving optimization and search problems where there's no known formula to find the optimal solution but where you can evaluate the "fitness" of potential solutions

2-BRUTE FORCE ALGORITHM

A brute force algorithm is a straightforward problem-solving technique that relies on exhaustive enumeration to find a solution. It systematically checks every possible candidate solution until the desired answer is found

GENETICS ALGORITHM

1. Initialization:
   - A population of candidate solutions (individuals) is randomly generated. Each individual represents a possible solution to the problem and is typically encoded as a chromosome (a string of values or characters).
2. Fitness Evaluation:
   - The fitness of each individual in the population is evaluated using a predefined fitness function. This function determines how good a particular solution is for the problem being optimized.
3. Selection:
   - A selection process chooses individuals from the current population to become parents for the next generation. Selection methods favor individuals with higher fitness scores, giving them a better chance of passing their genetic information (chromosomes) to the next generation. Common selection techniques include roulette wheel selection and tournament selection.
4. Reproduction:
   - Selected parent individuals undergo crossover, a process where they exchange parts of their chromosomes to create offspring (new candidate solutions). This simulates inheritance in biological reproduction. Crossover helps generate new combinations of existing solutions and explore the search space.
5. Mutation:
   - With a low probability, mutations are introduced into the offspring's chromosomes. This involves randomly changing some values in the chromosome. Mutation helps maintain diversity in the population and prevent premature convergence to a local optimum.
6. New Generation:
   - The offspring generated from crossover and mutation become the new population for the next iteration.
7. Termination:
   - The loop iterates through the steps above (2-6) for a predetermined number of generations or until a satisfactory solution is found.

BRUTE FORCE ALGORITHM

1.Define the Problem and Search Space:

- Clearly identify the problem you're trying to solve and all the possible candidate solutions (search space). This is crucial for brute force, as it needs to systematically evaluate every option within this space.

2. Iterative Evaluation:

- The core of a brute force approach lies in its iterative evaluation. The algorithm systematically loops through every single possible solution in the search space.

3.Solution Checking:

- At each iteration, the algorithm checks the current candidate solution against the problem's criteria. This might involve evaluating a fitness function or comparing it to a desired target value.

4. Solution Identification:

- If the evaluated solution meets the problem's criteria (e.g., it's the maximum value, the correct password, or the decryption key), the algorithm stops. This is the desired solution.

5. Exhaustive Search:

- If no solution is found in a particular iteration, the algorithm continues looping and evaluating the next candidate solution in the search space. This exhaustive search ensures all possibilities are considered.

6.Termination:

  - The algorithm terminates when either:A solution is identified that meets the criteria.
  - All possibilities in the search space have been evaluated (no solution found).

DEVELOPMENT PLATFORM

1. Jupyter Notebook:
   - Pros: Easy to use, interactive, good for data analysis and experimentation.
   - Cons: Not ideal for building production-ready applications.

2. Python Programming Language:
   - Pros: Beginner-Friendly, Large and Supportive Community, Versatile Language , Extensive Libraries and Frameworks , Readability and Maintainability , Cross-Platform .
   - Cons: Slower Execution Speed , Dynamic Typing , Memory Management, Limited Mobile Development.

3. Libraries Used In The Project:
   - Random , Math , Pandas , Matplotlib , Numpy .

LIBRARIES USED IN THE PROJECT

5. Random :
   - The random library in Python provides functions for generating pseudo-random numbers. These numbers are not truly random, but they are good enough for many applications.

2. Math:
   - The math library in Python provides a collection of functions for performing mathematical operations and constants.

3. Pandas :
   - Pandas is a powerful and popular open-source Python library specifically designed for data manipulation and analysis.

4. Matplotlib :
   - Matplotlib is a fundamental library in Python for creating static, animated, and interactive visualizations of data.

5. Numpy :
   - NumPy (Numerical Python) is a fundamental library in Python for scientific computing.

DATASET USED

1.Traveling Salesman Problem data set:

- We'll assume the salesperson lives in a boring, flat, 2D Cartesian plane, and that each city can be described simply as existing at a single (x, y) location. Each datafile describes some number of cities, one city per line in the file, where each city has a index (first city is index-0) and a location.

DATASET LINK:

https://www.kaggle.com/datasets/mexwell/traveling-salesman-problem

PSO ALGORITHM

1. Initialization:
   - A population of particles (potential solutions) is created. Each particle represents a candidate solution to the problem being optimized.
   - Each particle is assigned a random position in the search space, which defines the candidate solution it represents.
   - The particles are also assigned an initial velocity, which determines the direction and magnitude of their movement in the search space.
2. Fitness Evaluation:
   - Each particle's fitness is evaluated based on a predefined objective function. This function measures how good a particular solution is.
3. Personal Best Update:
   - Each particle keeps track of its own best position (personal best) encountered so far, based on its fitness evaluation.
4. Global Best Update:
   - The entire swarm also keeps track of the best position found by any particle in the population (global best).
5. Velocity Update:
   - The velocity of each particle is updated based on its current velocity, its attraction towards its personal best position, and its attraction towards the global best position. The update typically involves:Inertia: A term that encourages particles to continue moving in the same direction.
   - Cognitive Component: A term that pulls the particle towards its personal best position.
   - Social Component: A term that pulls the particle towards the global best position.
6. Movement:
   - Based on the updated velocity, each particle moves to a new position in the search space.
7. Iteration:
   - Steps 2 to 6 are repeated for a predetermined number of iterations.
8. Termination:
   - The algorithm terminates after reaching a maximum number of iterations or when a satisfactory solution is found.

RESOURCES PAGE

RESEARCH PAPERS LINKS:

1.Particle Swarm Optimization with Simulated Annealing for TSP(LUPING FANG, PAN CHEN, SHIHUA LIU):

- https://www.researchgate.net/publication/254456929_Particle_Swarm_Optimization_with_Simulated_Annealing_for_TSP

2.Literature Review on Travelling Salesman Problem(Chetna Dahiya & Shabnam Sangwan) :

- https://www.researchgate.net/publication/341371861_Literature_Review_on_Travelling_Salesman_Problem

3.Fuzzy Particle Swarm Optimization with Simulated Annealing and Neighborhood Information Communication for Solving TSP(Rehab F. Abdel-Kader) :

- https://www.researchgate.net/publication/215498887_Fuzzy_Particle_Swarm_Optimization_with_Simulated_Annealing_and_Neighborhood_Information_Communication_for_Solving_TSP

4.What is Simulated Annealing?(MICHAEL W. TROSSET) :

- https://www.academia.edu/48488662/What_is_Simulated_Annealing

LINKS FOR VIDEOS:

1.OR2 lecture 30 Simulated Annealing Algorithm:

- https://www.youtube.com/watch?v=NbVtS_1f2T4&t=252s

2.Particle Swarm Optimization Explained and Implementation in Matlab Step by Step:

- https://www.youtube.com/watch?v=nEYWrLiKfYY&t=1826s

THANK YOU!