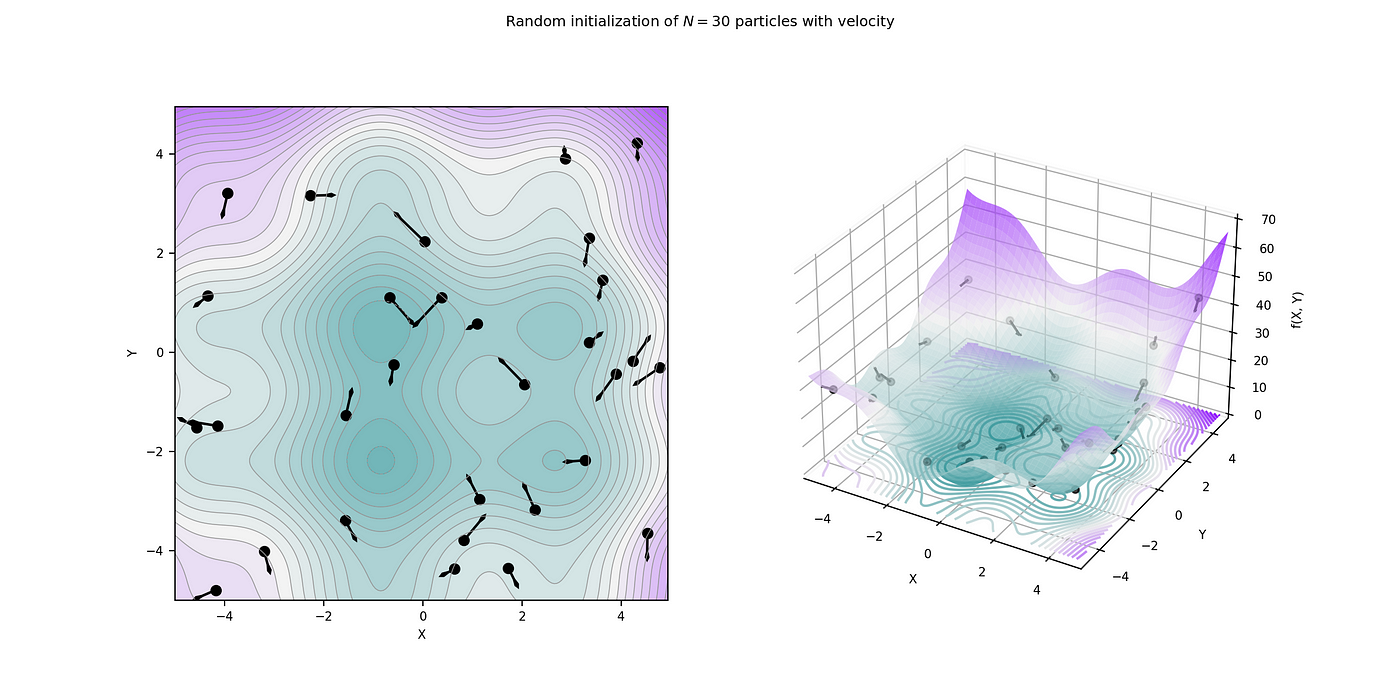
Group Algorithm: Particle Swarm Optimization



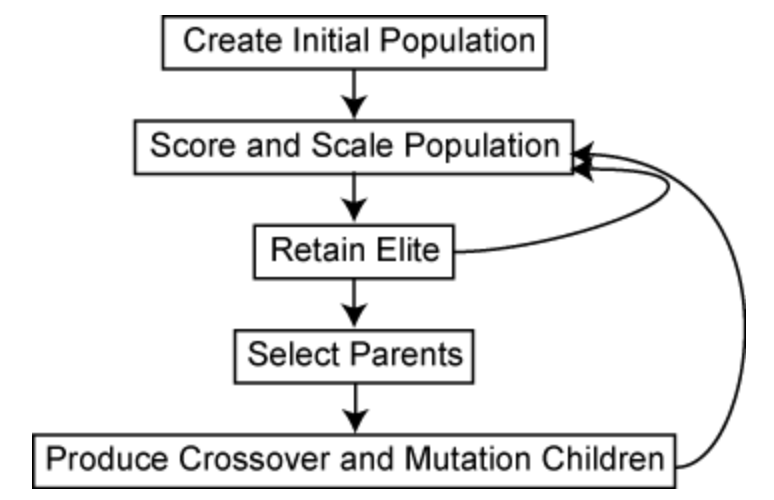
**Introduction:**

Particle swarm Optimization is an algorithm inspired by swarms of birds and fish in the wild. It was discovered by James Kennedy and Russel Eberhart in 1996. Originally the algorithm was written to imitate social behavior, but with some edits and simplification, it just turned into an optimization algorithm. PSO is what computer scientists call metaheuristic meaning it makes few to no assumptions on the problem it's given to optimize while it is still going over large amounts of potential solutions.

In PSO the solutions are called particles, and the particles move around in a search space. The particles move around using a few formulas based on their velocity and position in reference to the particle’s best option from its own movements as well as global movements from other particles. The use of global and local references makes the particles move in a swarm-like movement toward the best solution. The process is repeated with the hope of finding an optional solution.

While the PSO may be seen as rudimentary compared to other algorithms, it serves a very unique purpose just like linear and logistic regression that play a role due to their simplicity and easy understandability. The PSO's ability to efficiently search a large, nonlinear space makes it well-suited for complex optimization problems where more advanced algorithms would be intractable or get stuck in local minima. Its stochastic nature also allows the PSO to explore the search space more thoroughly than gradient-based methods.

**Comparison With Other Algorithms:**

PSO and Genetic algorithms (GA) are both optimization algorithms but have different methods of getting to the solution. From a discovery perspective, they were both inspired by nature PSO by swarms of birds swarming towards a common goal and GA was inspired by natural selection and genetics in nature. The two ways these algorithms get to the solutions is different though. On one hand, PSO uses swarm knowledge and individual knowledge of the particle to help find the solution. So over a vast amount of data, we can find a solution quicker with the knowledge of the overall swarm and also having the particle's thinking to what the solution may be. This works well because if the swarm is having better luck it can override a particle movement and vice versa. GA works in 5 steps with 4 being in a loop as shown in the image: As the image shows the population of possible solutions goes through a loop. If the loop finds a solution it should keep it, if they aren't a solution then more optimal possible solutions are paired and “bred”. Another aspect of this “breeding” is mutations, every time 2 organisms make a child they have a child that is similar but not identical to their set of genes (obviously the genes from each parent are the same). Basically, this algorithm has a population evolving over generations until they find the perfect population. It is a very interesting algorithm that makes you think about what other algorithms are made from nature. In terms of speed PSO is a safer bet as it converges from all angles, while GA may require many more generations. But both have different specialties so it's better to run both to see which algorithm better suits a specific problem. For problems with more complex problems GA might be the better option with its option to run massive population sizes. On the other hand if it's less complex but has a large space to cover PSO is a better option because of the swarm.

**Justification of Superiority:**

While PSO is not always the best algorithm to use, it most definitely has strong advantages in applications that need simple, lightweight machine-learning algorithms providing a fairly good solution in almost every case as it is not as susceptible to local minima as other ML algorithms like multilayered perceptron and logistical regressions. We can break this down by a

1. Simplicity and Ease of Implementation: PSO's conceptual model is simple and easy to understand, drawing from natural phenomena of bird flocking or fish schooling, primarily depends on fewer parameters than other similar algorithms (like cognitive and social components, inertia weight).

2. Efficiency in Convergence: PSO generally converges to a solution more rapidly than many other algorithms, making it efficient for time-sensitive applications. Through particle velocity adjustments, PSO effectively balances exploration (global search) and exploitation (local search), reducing the risk of being trapped in local optima that many other algorithms like MLP, logistic regression, and other

3. Robustness and Reliability: PSO demonstrates robust performance even in noisy and dynamic environments, making it reliable for real-world applications. Its ability to navigate complex, non-linear search spaces makes PSO particularly useful in real-world scenarios where such conditions are common. PSO can work for any number of datapoints and remains lightweight and robust with its global exploration capabilities.

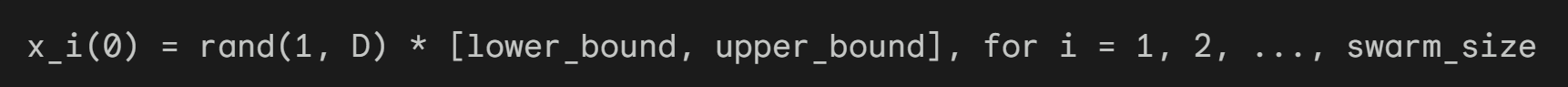
4. Flexibility: PSO's flexible framework allows for hybridization with other algorithms (like GA or Simulated Annealing) to enhance performance and overcome individual limitations. For example, a hybridized PSO with GA was shown to improve the quality of solutions for the TSP, a classic combinatorial optimization problem as PSO excels at global exploration, efficiently traversing the search space to identify promising regions, while GA excels at local exploitation, effectively refining solutions within those regions. PSO also strikes an effective balance between local and global search thanks to its use of both personal and neighborhood best solutions.

**Detailed Algorithm Explanation:**

Prior to initialization, the number of particles and the size of the search space is determined. Additionally, a cognitive and social parameter as well as an inertia weight is determined. The cognitive and social parameters control how much influence a given particle's own best position and the best known position of the entire swarm has over its velocity. The inertia weight scales the magnitude of the velocity. This affects how much a given particle is able to explore the search space.

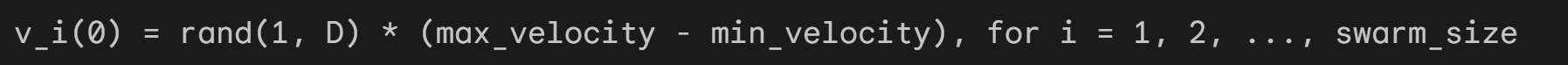
During initialization, particles are randomly placed within the search space and given a random velocity. The initial positions and velocities of the particles can be represented mathematically as follows:

Initial Position:



1. **x\_i(0)** is the initial position vector of the i-th particle
2. **rand(1, D)** generates a random vector of size 1 x D, where D is the dimensionality of the problem
3. **lower\_bound** and **upper\_bound** are the vectors defining the lower and upper bounds of the search space, respectively

Initial Velocities:



1. **v\_i(0)** is the initial velocity vector of the i-th particle
2. **rand(1, D)** generates a random vector of size 1 x D
3. **max\_velocity** and **min\_velocity** are the maximum and minimum allowable velocities, respectively

These initial positions and velocities set the stage for the iterative optimization process of the PSO algorithm. As the particles move through the search space, their positions and velocities are updated based on the best position they find and the best positions found by their neighbors. This collaborative approach allows the swarm to converge towards the optimal solution using collective knowledge, which is the primary advantage of PSO and the primary reason it can be both lightweight and not be easily stuck in local minima as we have multiple searchers instead of

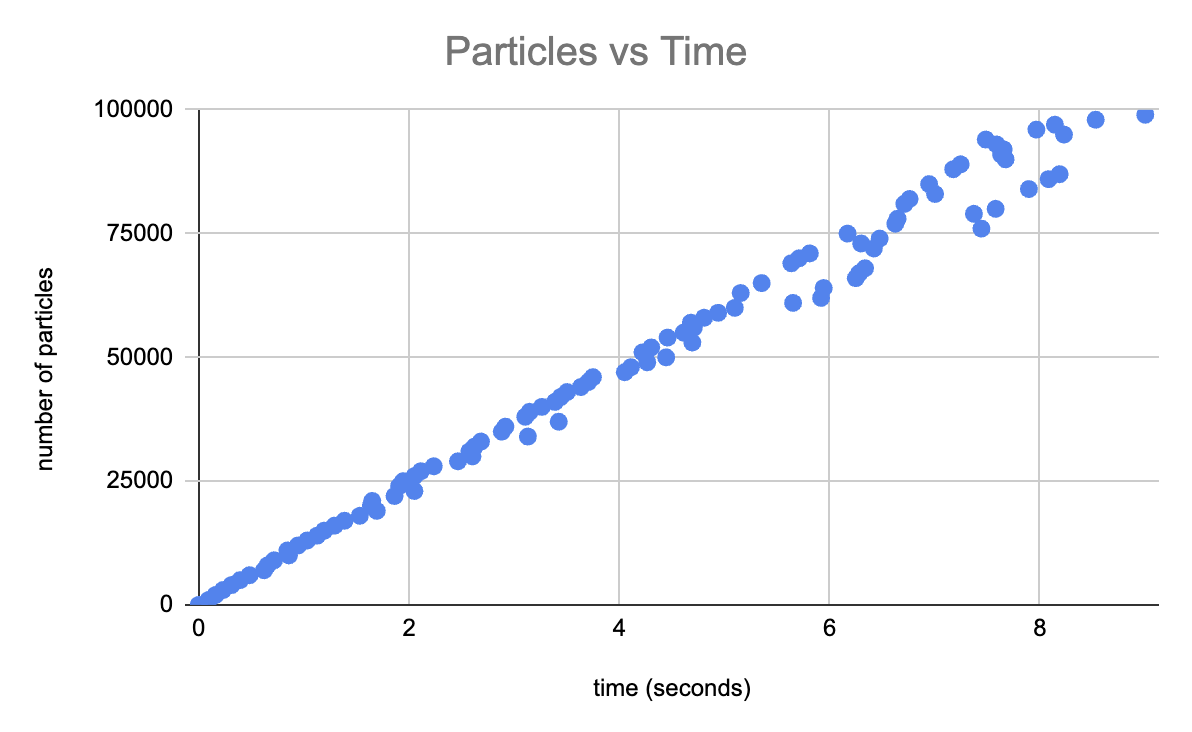
**Implementation Details:**

When implementing PSO in python, we first defined a Particle class. Each instance of this class contained a particle’s position, velocity, and best position. We defined the initial parameters c1, c2, and w in the global scope. We decided to implement PSO in two dimensions, or for a function with two input parameters, because this is simplest to visualize. It should be noted, however, that PSO can be implemented for functions with an arbitrary number of inputs. We then defined a function pso to perform the operations necessary for PSO. The function had inputs num\_particles, max\_iterations, and search\_space.

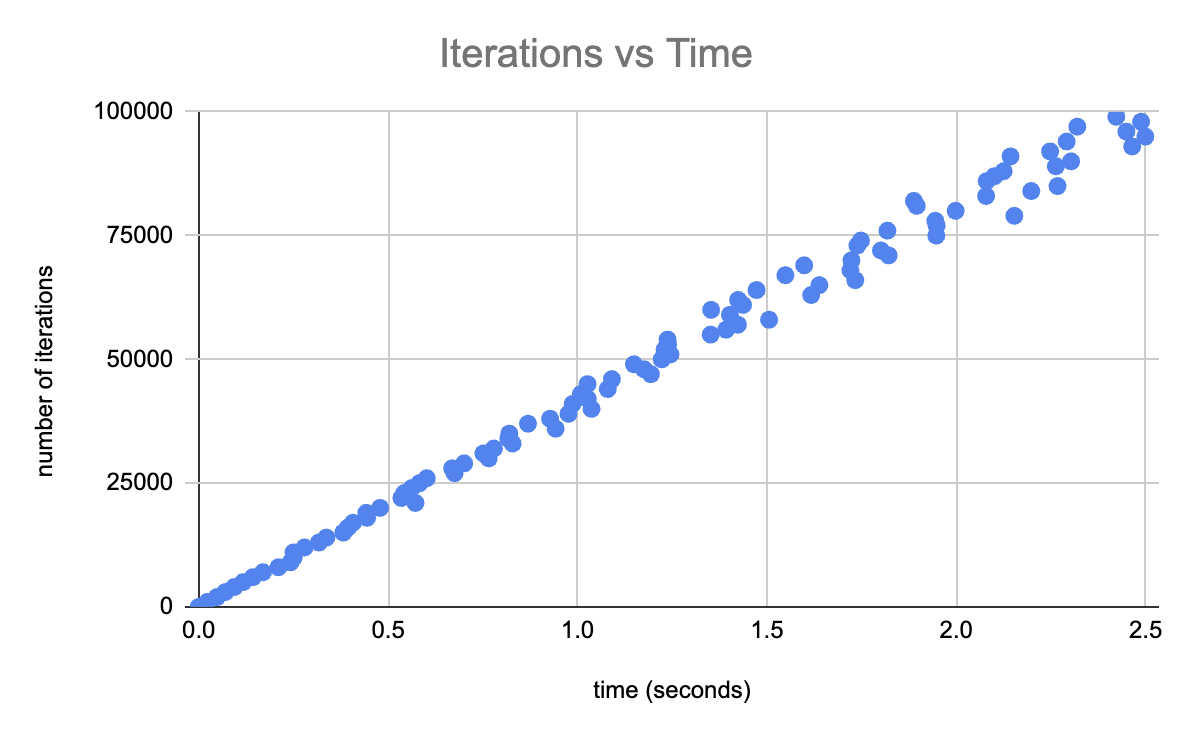
Within this function, a list of instances of the Particle class was initialized. This list would serve as an iterable within the main body of the pso function. We defined two variables global\_best\_position and global\_best\_value within the scope of the function to be referenced during iteration. We then iterated through the list of Particle instances max\_iteration times, updating each particle’s velocity, position, and best position. If the current particle’s best position was better than the global best position then the global best position was updated. Once iteration was complete, the optimized global position and value were returned.

To perform time complexity analysis, we wrote a script PSO-runtime-analysis.py which imported the main PSO program and ran pso.py with various input sizes. Since the algorithm is iterative the time complexity depended exclusively on the maximum number of iterations and the number of particles. For both of these variables, time complexity analysis was conducted. This consisted of keeping all variables but the one being tested constant and running pso for various values of the variable being tested, keeping track of the time required to run the algorithm. The time and corresponding input size was concatenated and appended to a csv file to later be imported to graphical software.

**Results and Analysis:**



*Figure 1: Time complexity analysis for variable particles.*



*Figure 2: Time complexity analysis for variable iterations.*

**Time Complexity Analysis:**

The time complexity of Particle Swarm Optimization (PSO) is primarily determined by the number of particles in the swarm (n\_particles) and the number of iterations in the optimization process (n\_iterations). The overall time complexity can be expressed as: **O(n\_particles \* n\_iterations).**

This can be further broken down into the following components: [A] **Evaluating the objective function for all particles,** [B] **Updating particle velocities,** [C]**Updating particle positions.** All 3 are O(n\_particles \* n\_iterations). [A] This operation involves calculating the value of the objective function for each particle in the swarm at each iteration. The objective function is a measure of how good a given particle's position is in terms of the problem being solved. Evaluating the objective function is typically the most computationally intensive part of the PSO algorithm, as it may involve complex calculations or simulations. For each particle, we have to fetch the current position of the particle, evaluate the objective function at the particle's position,

and store the result of the objective function evaluation.

Since these steps need to be performed for each particle at each iteration, the overall time complexity of this operation is O(n\_particles \* n\_iterations).

The dominant factor in the time complexity of PSO is the number of particles and iterations. This means that the time it takes to run PSO increases linearly with the number of particles and iterations. For [B] have to Calculate the velocity update vector based on the particle's personal best, global best, and current velocity and apply the velocity update vector to the particle's current velocity. For [C] we have to add the particle's updated velocity to its current position and

apply boundary constraints to ensure that the particle's new position remains within the search space. This is all O(1) so everything is O(n\_particles \* n\_iterations).

In practice, the number of particles and iterations is typically chosen based on the specific problem being solved. For example, problems with a large number of dimensions may require more particles to effectively explore the search space. Similarly, problems with complex landscapes may require more iterations to converge to the optimal solution.

**References:**

Wikimedia Foundation. (2023, October 26). *Particle swarm optimization*. Wikipedia. https://en.wikipedia.org/wiki/Particle\_swarm\_optimization

J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, WA, Australia, 1995, pp. 1942-1948 vol.4, doi: 10.1109/ICNN.1995.488968.

Tam, A. (2021, October 11). *A gentle introduction to particle swarm optimization*. MachineLearningMastery.com. https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/