CITS4404 Team Project - Building AI Trading Bots

Group 6 - Part I

Chen, Zijian
22998691@student.uwa.edu.au

Dai, Ethan
23625929@student.uwa.edu.au

Ida Bagus Prawira Janottama Kn
23894575@student.uwa.edu.au

Nadeesha Adikari
24041382@student.uwa.edu.au

Su, Daniel
22965999@student.uwa.edu.au

Townshend, Nathan
22970882@student.uwa.edu.au

April 11, 2025

Contents

1 Introduction

2 Gravity Search Algorithm

The Gravitational Search Algorithm (GSA) [13] was introduced in 2009, developed as another option for non-linear optimisation. Where the majority of nature-based optimisation algorithms use multiple agents with some form of swarm intelligence to perform exploration of the search space and exploitation of found maximums, GSA uses multiple agents but with rules based on Newtonian gravity and motion.

In GSA, Gravitational forces are applied on and between agents in the search space, with the mass of an agent is based on its fitness compared to the other agents (a more fit agent is more massive). Agents are initially randomly distributed throughout the search space. The new velocity (and hence position) of an agent at each time step is calculated by randomly retaining a portion of the previous velocity, and applying any acceleration due to gravity. The force due to gravity and the total number of agents can be modified over time steps, for example to have lower initial gravity to promote early exploration, or to reduce the number of agents later on to simplify the system and promote exploitation. While Newtonian gravity has masses attracted inversely proportional to the square of their distance (with $F \propto \frac{1}{R^2}$), the original paper on GSA has the force inversely proportional to their distance ($F \propto \frac{1}{R}$). Different implementations and versions of the GSA algorithm use either R or R^2 - for example of the two papers discussed below, GGSA uses R, and EGSA uses R^2.

2.1 Motivation for improvements to GSA

While GSA performed better in accuracy and convergence speed on benchmarks than other popular algorithms of the time such as PSO and Real Genetic Algorithm [13], there are a number of issues that have lead to modified versions of GSA being proposed and utilised.

The two phases of these nature-inspired algorithms - exploration and exploitation - generally conflict, with increasing the performance of an algorithm in one phase generally weakening the performance in the other. To combat this, hybrid algorithms combining multiple optimisation algorithms have been tested, for example SA-PSO (see Section **??** of this report) that uses Simulated Annealing to improve the exploitation performance of the PSO algorithm. However, while combining multiple algorithms this way improves performance, it can come at the cost of considerable additional computation, potentially even greater than the sum of running each algorithm sequentially. GSA is an effective optimisation algorithm for large-scale problems, as it is able to perform effectively with a significantly smaller agent count than other algorithms such as PSO [3], however it is not exempt from the trade-off of exploration vs exploitation. Local search has been suggested as a way to improve its exploitation [13], but is also computationally expensive particularly in higher dimensional search spaces. Thus, improvements able to be made to the exploration and exploitation performance of algorithms with minimal additional computational power would be of interest for large-scale problems where it might be infeasible to use hybrid algorithms or local search.

2.2 Enhanced GSA

Optimal Power Flow (OPF) for electrical grids is a large-scale optimisation problem of obvious interest to reduce operational costs, network losses, and environmental pollution. It has a discontinuous, nonlinear, and very large solution space making it difficult or infeasible to utilise some optimisation algorithms. Enhanced GSA (EGSA) was proposed by Jahan and Amjady [3] in 2013 to improve on the exploration of GSA, particularly for a problem with such a large solution space. The key modification in EGSA is the introduction of a 'mutation' operator, which continuously generates new agents throughout the solution space. When a mutation is performed, up to half of the worst agents are randomly selected, and a random number of their variables (position within a particular dimension) are changed to random values. Initially the random values can be from anywhere in that dimension to assist with exploring the entire solution space early on, with the size of the possible change being restricted and reduced as the number of iterations increase to support convergence later on in the search.

To evaluate the proposed algorithm, the paper selected a number of standard OPF optimisation algorithms (Genetic Algorithm, Differential Evolution, PSO, and GSA). Each algorithm, along with EGSA, were then benchmarked using 5 sample problems - 4 standard but small OPF IEEE datasets, and an additional less standard but significantly larger dataset. Each algorithm was given 100 agents and 1000 iterations. While GSA consistently performed well, EGSA outperformed all other algorithms on all sample problems - both with more optimal results, and requiring less compute time. Additionally, EGSA was the only algorithm that found a solution for the large dataset that was actually economically viable to utilise.

2.3 Gbest-guided GSA

As the number of iterations of GSA increase agents all increase in mass, therefore slowing down the overall search and exploitation due to increased inertia. GSA can additionally struggle during exploitation when a number of agents in a neighbourhood are attracted to each other and not necessarily able to accelerate towards a nearby optimum. Mirjalili and Lewis [10] proposed gbest-guided GSA (GGSA) in 2014 as a computationally cheap way to combat these issues and improve the exploitation of GSA.

GGSA addresses these two issues by introducing an additional force - a 'global best' solution is kept track of and updated each iteration, and agents are attracted to this 'gbest' solution in the same way they are attracted to the masses of the other agents. To prevent this from having a significant detrimental effect to early exploration, the gravitational force towards the other agents and the gravitational force towards the gbest solution are weighted separately, with the agent force prioritised during the exploration stage and dropping off as the gbest force is increasingly prioritised during the exploitation stage.

3 Modified Particle Swarm Optimisation with Simulated Annealing

3.1 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) was first introduced by Kennedy and Eberhart [9] in 1995 as a method for optimisation in continuous nonlinear search spaces. It was inspired by the swarming social behaviour exhibited naturally by species such as fish or birds. The motivation for mimicking nature in this particular way was the observation that individual members in flocks benefit from the collective experiences of all other members [15]. An example could be birds flocking to a food source, many individuals within the flock would have no prior knowledge of the location of a new food source but the information spreads to all individuals through flocking behaviour.

The original PSO algorithm operated on several basic rules for each individual within the swarm. Using some cost function, each individual remembered its own personal best (pbest) position and also knows the global best (gbest) position found by any individual within the swarm.

The velocity update of an individual depends on its distance relative to both pbest and gbest with hyperparameters p_increment and g_increment determining the magnitude of the velocity increase towards either point. The resulting velocity update is a vector addition of the velocity towards pbest and gbest.

3.2 Motivating Particle Swarm Optimisation with Simulated Annealing

Particle Swarm Optimisation with Simulated Annealing (SA-PSO) was introduced by Shieh et al. [14] to address two optimisation issues at the time of publishing.

1. Genetic Algorithms (GA) struggled with epistasis where genes were highly correlated with each other within the chromosome causing issues with crossover for new generations. It was also more computationally inefficient compared to PSO.
2. PSO struggled with premature convergence to local minima due to localisation of the particles. Due to the attraction to pbest and gbest, exploration is largely discouraged in the algorithm.

Simulated Annealing (SA) provides an exploration property using the metropolis process, which converges asymptomatically to the global optimum given certain preconditions. A novel SA and PSO hybrid approach is proposed due to the computational efficiency of PSO compared to other optimisation algorithms and the more stable convergence properties of SA stemming from the algorithm's greater ability to explore the search space.

A high initial temperature increases the acceptance probability for new (poorer) solutions, increasing the randomness of velocity updates leading to a larger subset of the search spacing being explored by particles. To stabilise the initial iterations, elitism (a concept from Genetic Algorithms where the top performing individuals are preserved into the next generation) for optimal solutions is maintained. As temperature drops, the swarm begins to converge more strongly onto gbest to perform exploitation of the known solutions in the search space.

3.3 Method and Results

The SA-PSO algorithm contains 6 hyperparameters that require tuning before being applied to the benchmark functions. This is performed through a coarse exhaustive search of all potential values within a range. For example, the initial temperature value was tested for between 50 and 90 in increments of 10.

The proposed SA-PSO algorithm and baseline GA, SA and PSO algorithms are then benchmarked on a selection of test functions with various dimensionality mostly taken from mathematics and physics such as the Zakharov function. The indicator used for performance is the rate of obtaining the optimal solution.

Each algorithm was run on each benchmark function 100 times. The final performance metric comparing SA-PSO to its predecessor algorithms is the average rate of obtaining the optimal solution.

The new algorithm demonstrated a significant performance improvement over the second-best algorithm PSO at an average convergence rate of 98.7% compared to 92.5%.

The consistency of the algorithms were also demonstrated through the mean value of the solution obtained over the 100 runs for each benchmark function. SA-PSO demonstrated means which were closest to the known optimum on the vast majority of benchmarks demonstrating the stability of convergence of the algorithm.

3.4 Discussion

The paper concluded that SA-PSO demonstrated strong results as a consequence of combining the explorative nature of SA to counteract the tendency of PSO to fall into local extrema. The performance justifies the claim that that SA-PSO is a capable optimisation algorithm for non-linear optimisation problems. However, the benchmark functions were all continuous and well-defined functions. The performance of SA-PSO on real-world data cannot be extrapolated from this paper as it may not be differentiable or continuous.

4 Grey Wolf Optimizer

The Grey Wolf Optimizer (GWO) aims to overcome key limitations in existing optimization algorithms, such as getting trapped in local optima, poor convergence behaviour, and an imbalance between exploration and exploitation. While algorithms like Particle Swarm Optimization (PSO), Gravitational Search Algorithm (GSA), and Differential Evolution (DE) have been successful in many scenarios, they often struggle with these issues. GWO addresses them by mimicking the social hierarchy and hunting strategies of grey wolves, leading to a more adaptive and balanced search process. The algorithm's structure, based on four types of wolves: alpha (α), beta (β), delta (δ), and omega (ω), enhances its ability to explore the search space globally and exploit promising regions effectively[11].

4.1 Limitations of Previous Approaches

The paper [11] highlights that metaheuristic techniques tend to outperform classical heuristics due to their simplicity, ability to avoid local optima, flexibility across different problem domains without structural changes, and their derivative-free approach that enables the process to begin from a random solution. Moreover, population-based algorithms have shown significant advantages over single-solution-based approaches.

GWO is a metaheuristic, population-based, swarm intelligence algorithm inspired by the social hierarchy and hunting behaviour of grey wolves. However, many existing algorithms still face critical challenges. A major issue is local optima trapping, where the algorithm prematurely settles on suboptimal solutions. Another significant challenge is the lack of balance between exploration and exploitation, some algorithms overexploit early good solutions and fail to adequately explore the broader search space, especially due to their stochastic behaviour. Furthermore, slow or improper convergence is also a limitation, as some algorithms struggle to efficiently reach high-quality solutions due to weak search strategies.These limitations have motivated the development of the GWO, which seeks to address these deficiencies through a more effective and biologically inspired approach.

4.2 Introduction of the Novel Idea

Although many earlier swarm intelligence (SI) algorithms were inspired by natural hunting or search behaviors, they often overlooked the internal leadership dynamics within animal groups. The Grey Wolf Optimizer (GWO) introduces a novel idea by simulating the leadership hierarchy and social behaviour of grey wolves. It categorizes the population into four types of wolves, alpha (α), beta (β), delta (δ), and omega (ω), to represent the hierarchy of decision-making. The alpha wolf represents the current best solution, the beta and delta are the second and third best, while the remaining candidates are considered omega wolves.

The algorithm also mimics the three main phases of hunting: searching for prey, encircling prey, and attacking prey. These behaviors guide the population's movement in the search space, with the alpha, beta, and delta wolves influencing the search direction. This structured yet dynamic mechanism allows the GWO algorithm to maintain a high degree of randomness and diversity, helping it to avoid local optima. [5]

4.3 Demonstration of the New Approach

The new approach is demonstrated through benchmarking experiments on 29 test functions. Among these, 23 are classical benchmark functions, and the remaining 6 are composite benchmark functions. These functions are used to evaluate the algorithm's performance in terms of exploration, exploitation, and local optima avoidance. Unimodal functions are used for exploitation analysis, multimodal functions for exploration analysis, and composite benchmark functions for local minima avoidance analysis. The algorithm was run 30 times for each benchmark, and statistical results (average and standard deviation) were compared to other popular swarm intelligence algorithms. Additionally, the GWO algorithm is applied to three classical engineering design problems: tension design, welded beam design, and pressure vessel design. It was also tested on a real-world optical buffer design problem, running the algorithm 20 times to validate its ability to solve real-world problems with constraints. [12].

4.4 Results and Validation

The performance of GWO was validated through benchmarking against well-known optimization algorithms, such as PSO, GSA, and GA, across various test functions, including unimodal, multimodal, and composite functions. Statistically significant improvements were observed in key metrics such as Delay-Bandwidth Product (DBP) and Normalized DBP (NDBP), with GWO achieving a 93% improvement in bandwidth and a 65% improvement in NDBP compared to existing methods. Additionally, the algorithm was applied to a real-world optical buffer design problem, where it demonstrated superior performance.The results were validated through multiple runs to ensure consistency, and the algorithm was tested on a high-performance computing (HPC) cluster using multiple CPUs, validating its robustness in solving large-scale, complex problems.

4.5 Assessment of the Conclusions

The conclusions presented in the paper are generally well-supported by the experimental results. The authors demonstrate that GWO performs competitively against established optimization algorithms across a wide range of benchmark functions and real-world problems. The balance between exploration and exploitation is effectively illustrated, and the algorithm shows strong potential in avoiding local optima and achieving reliable convergence. However, while the findings are promising, broader testing on diverse problem domains would help further validate the general applicability of the approach.

5 Harris Hawks Optimisation

The paper presents a new algorithm called Harris Hawks Optimiser, known as HHO [1]. It aims to solve difficult optimisation problems. Many real-world problems are complex, they might involve many variables, have many possible good solutions which is called multimodal behaviour, change suddenly meaning they are non-differentiable, or have strict rules also known as constraints. Old methods, like standard mathematical techniques, often find these problems hard to solve. Newer algorithms, called metaheuristics like Genetic Algorithms or Particle Swarm Optimisation, are easier to use because they don't need complex maths. However, these newer algorithms also have problems. They can be sensitive to their settings, a process called parameter tuning, and sometimes get stuck on a solution that is good but not the best, known as a local optima. Also, a rule called the "No Free Lunch" theorem suggests no single algorithm is perfect for all problems. This encourages scientists to create new optimisation algorithms. HHO is a new algorithm, inspired by nature, designed to be a good option for solving these complex problems [1].

Older algorithms often failed because they were not efficient for the complex problems found in the real world. Even the newer metaheuristic algorithms can fail. They might find a quite good solution quickly but then get stuck there, called premature convergence or local optima stagnation. This happens if they don't do the exploration (balance searching widely for different solutions) and searching deeply near good solutions, known as exploitation. Also, their success can depend a lot on choosing the right settings, which can be difficult. The idea that no single algorithm works best for everything also motivates looking for new approaches like HHO [1].

HHO is new because it copies the teamwork and hunting style of Harris' hawks, especially their "surprise pounce" attack. The algorithm uses maths to copy how these smart birds hunt together. New ideas in HHO include copying how hawks work together to attack based on what the prey does, called the Teamwork Model. It uses a changing "escaping energy" number, represented as E, for the prey, which helps the algorithm switch smoothly from searching widely, the exploration phase, to searching deeply, the exploitation phase, as it runs, called the Changing Strategy. It has four different plans for the exploitation phase, chosen based on the prey's energy E and its chance of escaping, represented as r, copying how hawks change tactics, called Different Attack Plans. In some attack plans, HHO uses a special random walk called Levy flight, or LF, to copy the tricky, sudden movements of prey and the quick dives of the hawks, helping to search better in a local area. Also, hawks in the algorithm only move to a new position if it is better than their current one, which helps make the solutions better over time (Choosing Better Moves). These ideas aim to help HHO balance searching widely and searching deeply, avoid getting stuck on bad solutions, and find the best overall solution [1].

The paper explains the HHO algorithm using mathematical formulas. It describes the exploration phase, which involves searching widely, and the four different exploitation phases, which involve searching deeply using different plans based on prey energy and escape chance. Each phase has its own maths equation telling the hawks how to move. The authors tested how well HHO works by trying it on many problems. They used 29 standard test problems often used to check optimisation algorithms. These included simple problems known as unimodal, problems with many good-but-not-best solutions known as multimodal, and very complex mixed problems called composition problems. They tested if HHO still works well when the problems become very large, having high dimensions up to 1000 variables. They also used HHO to solve six real engineering design problems that have strict rules or constraints. The paper includes a pseudocode for the algorithm, and the authors say the computer code is available online, making it easier for others to check or use it [1].

HHO was compared to 11 other popular optimisation algorithms. The comparison looked at the best, worst, average, and consistency, measured by standard deviation, of the results over 30 tries for each problem. They also looked at how the algorithm behaved while searching, known as qualitative results. They tested how well HHO handled problems of increasing size, which is called scalability testing. A statistical test, the Wilcoxon rank-sum test, was used to see if HHO's better results were truly significant. The results showed HHO performed much better or was very competitive compared to the other algorithms on most test problems, including the very large ones. It was good at finding the best solutions and didn't slow down as much as others when problems got bigger. HHO also found the best or very good solutions for the real engineering problems. The statistical tests confirmed that HHO's improvements were meaningful in many cases [1].

The authors conclude that HHO is a very good optimisation algorithm. They state it gives excellent or competitive results compared to many well-known algorithms on both standard test problems and real engineering problems. They believe its success comes from its special mix of search strategies copied from Harris' hawks, like the changing energy level, the different attack plans using Levy flights, and only choosing better moves. They tested HHO thoroughly on many different types of problems, compared it fairly to many other algorithms, and used statistics to check the results. The results consistently show HHO performing very well, handling large problems effectively, and finding good solutions for real-world tasks. The paper provides strong evidence that HHO is a useful new optimisation algorithm [1].

6 Artificial Bee Colony

6.1 Background and Motivation

The Artificial Bee Colony (ABC) algorithm was first introduced by Dervis Karaboga in 2005. It was developed to address limitations observed in existing optimization algorithms, such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), particularly their tendencies toward premature convergence and inadequate exploration of complex, high-dimensional search spaces.[6]

6.2 Novelty and Improvements

ABC's uniqueness stems from its emulation of the natural foraging behaviour of honeybee swarms. The algorithm assigns bees into three distinct roles:[6][7]

1. Employed Bees: These bees exploit known food sources, representing current candidate solutions.
2. Onlooker Bees: They assess the quality of food sources based on information shared by employed bees and probabilistically choose sources to explore further.
3. Scout Bees: They conduct random searches to discover new food sources, aiding in escaping local optima and enhancing global exploration.

This division of labour enables ABC to balance exploration and exploitation effectively, mitigating the risk of premature convergence.

6.3 Core Concept and Mechanism

The operational framework of the ABC algorithm involves iterative cycles comprising:

1. Initialization Phase: Random generation of an initial population of candidate solutions.
2. Employed Bee Phase: Each employed bee modifies its current solution based on a neighbourhood search and evaluates the nectar amount (fitness) of the new solution.
3. Onlooker Bee Phase: Onlooker bees select food sources based on the quality information shared by employed bees and further exploit these sources.
4. Scout Bee Phase: If a solution is not improved over a predetermined number of cycles, it is abandoned, and the corresponding employed bee becomes a scout, randomly generating a new solution.

This process repeats until a termination criterion, such as a maximum number of iterations or a satisfactory fitness level, is met.[7]

6.4 Validation of Effectiveness

Karaboga and Basturk conducted extensive evaluations of the ABC algorithm using benchmark optimization functions, comparing its performance against algorithms like GA, PSO, and Differential Evolution (DE). Their studies demonstrated that ABC outperformed these algorithms in terms of global optimization capability and robustness across various test scenarios.[6][7]

6.5 Conclusion and Assessment

The findings from these studies affirm that the ABC algorithm effectively addresses the shortcomings of earlier optimization methods. Its design, inspired by natural foraging behaviors, allows for a harmonious balance between exploration and exploitation, making it particularly adept at navigating complex, multimodal optimization landscapes. While the algorithm's performance may vary depending on specific problem characteristics, its overall adaptability and efficiency render it a valuable tool in the field of optimization.[8]

7 Firefly Algorithm

The Firefly Algorithm (FA) is a nature-inspired algorithm designed to address significant limitations in traditional global optimization methods, particularly in multimodal landscapes where multiple optimal solutions exist. Observing from the fireflies' behavior, the algorithm leverages attraction-based movement and local interaction to guide solution agents through the search space. Initially proposed by Xin-She Yang in 2008 [16], FA offers a generalized and flexible optimization framework that has also been adapted to real-world domains, such as in the design of dynamic pricing strategies for autonomous pricing bots [4].

7.1 Motivations

Traditional algorithms such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) struggle with multimodal landscapes. They tend to converge prematurely to local optima and often require elaborate tuning or hybridization to improve performance. GA relies on crossover and mutation mechanisms, which are often hard to tune and inefficient for fine-grained local search. PSO, although simpler and generally faster, tends to focus too heavily on the global best, causing particles to cluster prematurely and miss alternative optima [16]. Neither approach provides a natural way to simultaneously explore multiple promising regions.

7.2 Methods

FA is based on the flashing and attraction behavior of fireflies, particularly how real fireflies use light signals to attract mates and prey, and how these interactions naturally result in swarm-like movement and synchronization [16]. The algorithm formalizes this through three foundational principles:

Unisex Attraction: Every firefly is attracted to every other, removing pair matching constraints and simplifies interactions.

Attractiveness Linked to Brightness (Objective Value): A firefly's "brightness" is determined by the value of the objective function — the better the solution, the brighter it is. Fireflies move towards brighter ones.

Distance-Dependent Influence: Attractiveness and brightness decay with distance, following either an exponential or rational decay function, inspired by the physical attenuation of light over distance.

The movement rule integrates local exploitation (via attraction to brighter neighbors) and global exploration (via random perturbations), governed by tunable parameters. These parameters include γ, light absorption coefficient, controlling the visibility range, α, randomization factor, introducing diversity, and β, base attractiveness. Fireflies only respond to nearby, brighter fireflies, the swarm can split into multiple subgroups, each attracted to a different local optimum.

This means FA can locate multiple good solutions in a single run, and handle rugged fitness landscapes without extra diversity mechanisms.

7.3 Application and Results

The Firefly Algorithm's biological metaphor proves adaptable in applications to dynamic pricing [4]. In these environments, each seller is modeled as a firefly. It emits "flashes", a signal, when ready to update prices. Other sellers observe the signals and adjust their pricing strategies. This results in a decentralized form of coordination where sellers gradually align their price update intervals and magnitudes, without explicitly sharing strategies.

The proposed method is demonstrated through the following:

Mathematical modeling and proof of convergence of synchronized pricing: proof that sellers using this model will allow initially unsynchronized pricebots to synchronize their price steps in equilibrium, increase the probability of making the best offer to buyers in the marker, and converge more accurately but less rapidly than a dynamic pricing model without synchronization [4].

Simulation of a multi-agent online market environment with realistic settings, including multiple sellers, buyers, and product attributes.

Evaluation across various metrics: seller profit, market price stability, and convergence speed. Compared against Fixed Pricing, Myoptimal Pricing Strategy, Game-Theoretic Strategy, Goal-Directed Strategy, Derivative Following Pricing Strategy, and Minimax Regret Strategy.

Results show that sellers using FA algorithms consistently outperformed those using unsynchronized methods. FA strategies improved profits by 10%–78% in most scenarios and provided faster, smoother convergence to competitive pricing, and greater price stability. Validation is achieved through simulations with varying market sizes (3 to 5 sellers, 500 to 1000 buyers), randomized buyer preferences, and multiple competitor configurations. However, Synchronization improves accuracy but may reduce speed of convergence. In rare cases (3.55% or less), multiple FA sellers competing against each other marginally reduce performance slightly due to internal competition. Still, in most configurations, FA sellers dominate market share, even when competing against game-theoretic or well-informed heuristic strategies like Myoptimal and Goal-Directed.

7.4 Discussion

The paper [4] claims that FA outperforms traditional dynamic pricing models across a range of scenarios. It presents this as the first successful application of emergent synchronization models to price coordination. The conclusions drawn by the authors are well-supported by both theoretical and empirical evidence. They successfully demonstrate that biologically inspired synchronization can enhance the adaptiveness and profitability of dynamic pricing mechanisms in decentralized markets. Moreover, the authors are careful to acknowledge the limitations of their approach, such as the assumption of homogeneous seller behavior and the need for real-world testing.

Overall, the Firefly Algorithm represents a powerful and flexible approach to optimization, capable of handling diverse, multimodal, and high-dimensional problems with a simplicity that makes it appealing for real-world application. Its decentralized, self-organizing structure offers a natural advantage in distributed systems where coordination without explicit communication is needed.

8 Conclusion

Algorithm	**Date**	**Category**	**Sub-category**
GSA	2009	Physical Phenomena & Laws of Science	Universe
SA-PSO	2011	Swarm Intelligence	Social Behaviour
GWO	2014	Swarm Intelligence	Foraging
HHO	2019	Organism-based	Fauna
ABC	2007	Swarm Intelligence	Foraging
FA	2009	Swarm Intelligence	Social Behaviour

Table 1: Optimisation algorithms and their categories and sub-categories (according to Alexandros Tzanetos' Mendelay repository).

Algorithm	**Strategy**
GSA	Simulates an n-body gravitational pull based on the fitness of the candidate solutions for positional updates.
SA-PSO	Uses simulated annealing where an initial high temperature parameter increases the randomness of new positions for individuals.
GWO	Uses a hierachical system for determining velocity updates of individuals by considering the positions of the top n individuals and giving each a weight.
HHO	Adapts the population's behaviour from focusing on exploration to exploitation and vice versa based on the quality of the current solutions.
ABC	Splits the population into multiple classes with separate roles, one randomly explores the search space and others exploiting around the current best solution.
FA	Incorporates a distance metric into individual position updates where attractiveness of an individual to any other depends not only on the quality of the solution but the distance to the other individual preventing the global best from exerting an outsized influence on velocities especially at initialisation.

Table 2: Ways in which swarm algorithms prevent early convergence.

Many modern population-based optimisation algorithms are rooted in the principles underpinning two major algorithms emerging in the late 20th century; Genetic Algorithms first hypothesised by Turing, later implemented and popularised largely through Holland's work [2], and Particle Swarm Optimisation in 1995 by Kennedy et al. [9]. These early optimisation algorithms performed well on the problems with simple search spaces but tended to struggle on multimodal functions. Original GA and PSO had tendencies to fall into local extrema as they over-exploited their current locations in the search space.

The algorithms reviewed in this paper provide some perspective into the various ways PSO can be altered to address the issue of early convergence (Table 2). The common theme is that improved performance for swarm algorithms requires some enforced exploratory behaviour apart from hyperparameter tuning to counteract the inherent tendency for individuals to bee-line for the initial global best.

Two overarching methods were used to achieve further exploration:

1. Taking the top n solutions and weight their influence on all individual updates (GSA, GWO, HHO, FA).
2. Using some form of stochasticity to explore the search space (SA-PSO, ABC).

The algorithms were tested on similar but ultimately different functions so a direct comparison isn't possible. A possible future analysis of top n methods and stochastic exploration methods on especially multimodal and flat unimodal functions can reveal the extent of their exploratory capabilities. However, all algorithms performed better compared to classic PSO and GA in their respective results, especially on heavily multimodal functions.

References

[1] Ali Asghar Heidari et al. “Harris hawks optimization: Algorithm and applications”. In: *Future generation computer systems* 97 (2019), pp. 849–872.

[2] John H Holland. “Genetic algorithms”. In: *Scientific american* 267.1 (1992), pp. 66–73.

[3] Mohammad Sadegh Jahan and Nima Amjady. “Solution of large-scale security constrained optimal power flow by a new bi-level optimisation approach based on enhanced gravitational search algorithm”. eng. In: *IET generation, transmission & distribution* 7.12 (2013), pp. 1481–1491. ISSN: 1751-8687.

[4] Janyl Jumadinova and Prithviraj Dasgupta. “Firefly-inspired synchronization for improved dynamic pricing in online markets”. In: *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE. 2008, pp. 403–412.

[5] P Kandasamy. “Literature Review on Grey Wolf Optimization Techniques”. In: *Int. J. Sci. Res.(IJSR)* 9 (2020), pp. 1765–1769.

[6] Dervis Karaboga and Bahriye Basturk. “A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm”. In: *Journal of global optimization* 39 (2007), pp. 459–471.

[7] Dervis Karaboga and Bahriye Basturk. “Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems”. In: *International fuzzy systems association world congress*. Springer. 2007, pp. 789–798.

[8] Ebubekir Kaya et al. “A review on the studies employing artificial bee colony algorithm to solve combinatorial optimization problems”. In: *Engineering Applications of Artificial Intelligence* 115 (2022), p. 105311.

[9] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95-international conference on neural networks*. Vol. 4. ieee. 1995, pp. 1942–1948.

[10] Seyedali Mirjalili and Andrew Lewis. “Adaptive gbest-guided gravitational search algorithm”. eng. In: *Neural computing & applications* 25.7-8 (2014), pp. 1569–1584. ISSN: 0941-0643.

[11] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. “Grey wolf optimizer”. In: *Advances in engineering software* 69 (2014), pp. 46–61.

[12] Seyedali Mirjalili et al. “Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization”. In: *Expert systems with applications* 47 (2016), pp. 106–119.

[13] Esmat Rashedi, Hossein Nezamabadi-pour, and Saeid Saryazdi. “GSA: A Gravitational Search Algorithm”. eng. In: *Information sciences* 179.13 (2009), pp. 2232–2248. ISSN: 0020-0255.

[14] Horng-Lin Shieh, Cheng-Chien Kuo, and Chin-Ming Chiang. “Modified particle swarm optimization algorithm with simulated annealing behavior and its numerical verification”. In: *Applied Mathematics and Computation* 218.8 (2011), pp. 4365–4383.

[15] Edward O Wilson. *Sociobiology: The new synthesis*. Harvard University Press, 2000.

[16] Xin-She Yang. “Firefly algorithms for multimodal optimization”. In: *International symposium on stochastic algorithms*. Springer. 2009, pp. 169–178.