

Personal Code: kbf569

Computer Science Extended Essay

A Comparison of Genetic Algorithms and Particle Swarm Optimization Algorithms

Research Question:

How does the **genetic algorithm** compare to that of the **particle swarm optimization algorithm** in providing solutions to **optimization problems** in terms of **speed** and **accuracy**?

Word count: 3997

CS EE World

<https://cseeworld.wixsite.com/home>

November 2022

29/34

A

Submitter Info:

Name: Jack Cantwell

Email: jack [dot] cantwelljc [at] gmail.com

Table of Contents

INTRODUCTION	3
THEORY	4
OPTIMIZATION PROBLEMS	4
OPTIMIZATION ALGORITHMS	5
GENETIC ALGORITHMS	7
PARTICLE SWARM OPTIMIZATION ALGORITHMS	10
HYPOTHESIS & APPLIED THEORY	13
METHOD.....	13
INDEPENDENT VARIABLES	13
DEPENDENT VARIABLES.....	16
CONTROLLED VARIABLES	17
PROCEDURE	19
DATA PROCESSING.....	19
CONCLUSION	24
BIBLIOGRAPHY.....	25
APPENDICES	28
APPENDIX A: PROGRAMS USED IN THE EXPERIMENT	28
APPENDIX B: EXAMPLE CODE	30
APPENDIX C: RAW DATA	31

INTRODUCTION

This primary focus of this essay is to investigate different metaheuristic strategies for optimization, specifically genetic algorithms, and particle swarm optimization (PSO) algorithms. In computer science and mathematics, an optimization problem is a problem that involves finding the best solution out of all possible solutions (FrancQ, 2011). One method of solving an optimization problem involves a brute-force search (or exhaustive search) which involves listing all the possible solutions of a problem and searching for the “correct” or most optimal solution. However, for problems with a high time complexity, the brute-force approach becomes infeasible. Time complexity is the term used to refer to the amount of time taken by an algorithm to run given the amount of input values (Great Learning Team, 2022). Some optimization problems such as the famous travelling salesman problem (TSP), a problem in which one must minimise the total route distance between N number of cities, have a time complexity class of factorial time, meaning that for N number of cities, the total number of operations (possible routes) is N factorial (Chase, et al., Not dated). The problem takes too long and requires large amounts of computing resources to run, hence why some optimization problems are better solved using heuristics and metaheuristics. Heuristics are techniques that are adapted to the specific problem and are quicker than a brute-force search, such as the nearest neighbour method for the TSP where the salesman chooses the nearest unvisited city as the next move. A metaheuristic is a higher-level problem-independent method that can be applied to most optimization problems (Glover & Sörensen, 2015). Examples of metaheuristics are genetic algorithms and particle swarm optimization algorithms, among many others. What struck me as interesting about so many of these metaheuristics is the heavy inspiration from biology and how the field of computer science attempts to mimic nature. Genetic algorithms use biological processes such as natural selection, evolution, mutation, and gene crossover whereas particle swarm algorithms mimic swarm-like intelligence, such as birds hunting for food. These algorithms are used in various industries and are incredibly useful in optimizing a variety of things such as NASA using evolutionary algorithms to design a more efficient antenna (Lohn, et al., 2005). Hence comparing two algorithms to see which is superior and more suitable for use in the real world can lead to many discoveries in the field of science and is worthy of investigation. There are

two large branches of metaheuristics, one that mimics evolutionary mechanisms and one that mimics swarm-like behaviours. Choosing one algorithm from each category and comparing them would not only show which algorithm is better but could also suggest that a certain method (evolutionary or swarm intelligence) is better than the other. Two major factors of good metaheuristics are speed and accuracy, as the algorithm needs to be fast enough to be worth using but must also be accurate enough to provide an appropriate answer to the problem. All these factors gave light to the research question: “How does the **genetic algorithm** compare to that of the **particle swarm optimization algorithm** in solving **optimization problems** in terms of **efficiency** and **accuracy**?”. The criteria for efficiency and accuracy will be discussed further in the method section of the essay.

THEORY

Optimization Problems

An optimization problem is a problem that consists of finding the best solution out of all feasible solutions, a finite set of variables. Optimization problems can be split into two categories: combinatorial and continuous. Combinatorial optimization problems consist of solutions of only certain values with distinct spaces between values (e.g., whole numbers) whereas continuous optimization problems consist of a constant sequence of solutions and can take any value within a range. For each instance within a combinatorial optimization problem, the instance is defined by a pair (F, c) with F representing the search space and c representing the ‘cost’ or ‘fitness’ for each solution of F (for example with the travelling salesman problem the ‘cost’ would be the distance of the specific route, F). (FrancQ, 2011) The cost function or fitness function is also more commonly referred to as the objective. To optimise the problem, one typically is aiming to find the maximum or minimum of objective function, for example, in the TSP the aim is to find the shortest distance between the cities thus the objective is to find the lowest possible value of the objective function otherwise known as the minimum of the function.

Another distinction within optimization problems is unconstrained optimization and constrained optimization. The distinction arises between whether there are constraints on the variables or not, for example the constraints could simply be bounds on the variables such as $f(x) \geq 0$. These constraints can either be soft constraints, which simply have variables that are penalised in the objective function, or hard constraints, which are conditions that must be met (Wikipedia, 2022). A large range of characteristics and qualities of optimization problems will be looked at further when developing the test set for the experiment later in this essay.

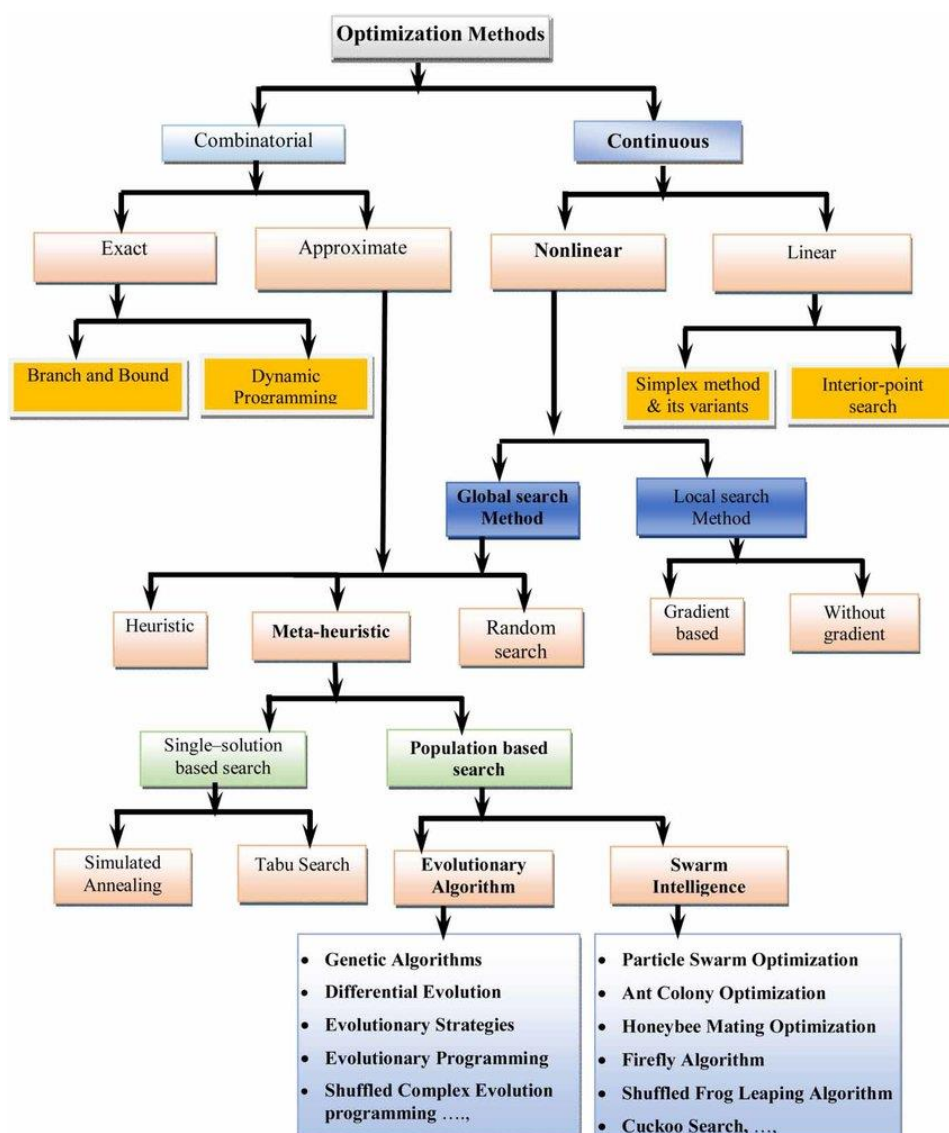
Optimization Algorithms

Optimization algorithms are iterative procedures that involve comparing various solutions until an exact or possible optimal solution to an optimization problem is found. (Mechanical Engineering at IIT Madras, No date).

Figure 1 categorises different optimization algorithms in different steps and arises new and relevant categorisations and methods of optimization algorithms such as **exact, approximate, linear and non-linear programming, global search, local search, population-based, and single-solution based**. Not all the terms mentioned in the graph are relevant to this essay, so only certain terms will be discussed however many of the terms relating to optimization problems are relevant as it is important to include a range of problems in the test set when evaluating the efficiency and accuracy of an optimization algorithms. Initially the categorisation is based on the optimization problem that the algorithm is being used to solve, whether it is a continuous or combinatorial (discrete) problem. If the problem is continuous, the next distinction is whether the problem is linear or non-linear. Linear programming is a process of solving an optimization problem where the constraints and objective function are both linear relationships (Luenberger & Ye, 2008) Whereas non-linear programming involves a problem where either objective and/or constraints consist of a non-linear relationship. Global search methods for continuous problems, along with approximate solutions to combinatorial problems lead to the choice of a heuristic, a meta-heuristic, or a random search to provide a solution. Out of the three options of heuristic, meta-heuristic and random search, meta-heuristic branches off into another classification dimension of single-solution or

population-based search. The, the single-solution based search focuses on improving a single candidate solution. The population-based method improves on multiple candidate solutions and is the focus of this essay. Genetic algorithms are found under the evolutionary algorithm category of population-based searches and particle swarm optimization algorithms are classified under swarm intelligence.

Figure 1: Classifying optimization algorithms based on aspects from the optimization problem being solved and the optimization method (Kumar, 2020)



Genetic Algorithms

Genetic Algorithms were created by John Holland in the 1960's, the goal of this was to study the concept of adaption in nature and develop ways of importing biological processes such as natural adaption into computer systems. (Mitchell, 1999)

As briefly touched on, a genetic algorithm is a metaheuristic, stochastic, global search optimization that falls under the category of Evolutionary Algorithms. Evolutionary algorithms are inspired by the mechanisms of biological evolution and have three main characteristics: population-based, fitness-oriented (fitness value given to each individual solution), and variation-driven (random changes in the solution from iteration to iteration). (Gad, 2018)

The processes of genetic algorithms can be distinguished into five phases:

1. Creating the initial population
2. Calculating the fitness of each solution
3. Selection
4. Gene crossover
5. Gene mutation

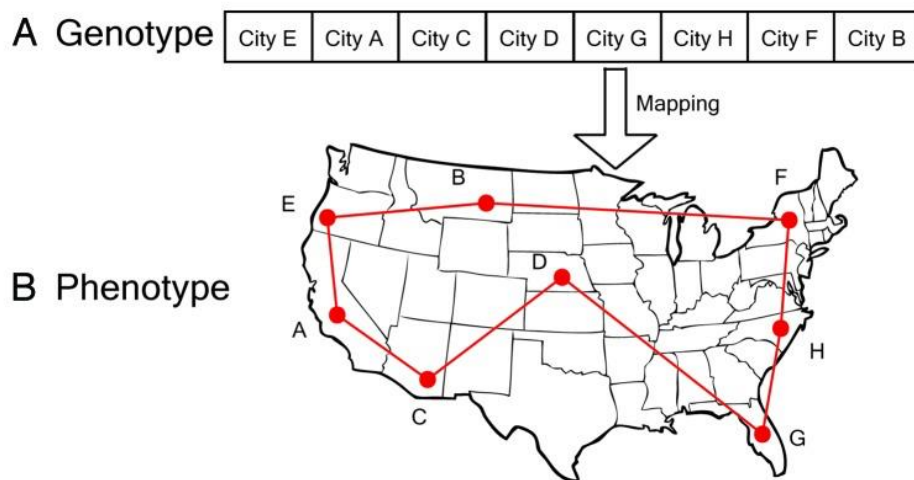
Creating the Initial Population

Creating the initial population for the genetic algorithm requires encoding the solutions into "chromosomes". Each individual candidate needs to be encoded into strings of numerical values, commonly the solutions are encoded into binary (0s and 1s) and the solution is comprised of an array of parameter values called "genes" and these strings of genes are what forms the chromosomes. (Dutta, 2021) The chromosome can also be referred as the genotype which is defined as the set of instructions to be decoded to form the phenotype which is used as an evaluable solution. (Manning, et al., 2012). Figure 2 shows a possible genotype and phenotype the travelling salesman problem, each city in the array of parameters is a gene and the arrangement of genes forms the chromosome or genotype, this can be mapped to create a phenotype and presented in a form that is easily evaluable. If the optimization problem is continuous, the 'chromosomes' for each solution is generated by the position in the search.

E.g., if the problem were to have 2 dimensions, x and y , then the 'chromosome' for a solution could look like $x, y = [49.35, 89.57]$.

The initial population is comprised of a set of chromosomes of randomly assigned values within the specified search space, this forms the first generation of solutions.

Figure 2: **A possible genotype and phenotype representation for the TSP.** (Manning, et al., 2012)



Calculating the fitness of each solution

Each candidate solution is evaluated by the fitness function, the function that the algorithm is attempting to optimise, and gives a measure to how close a given solution is to achieving the optimum. At each iteration or generation, each solution in that iteration is assigned a fitness or cost value. In optimization problems where there are constraints (such as an inequality equation of $x \geq 0$), sometimes a penalty method is used when a solution. This involves multiplying the cost value by a penalty parameter that is calculated by measuring the extent of the breach of the constraint (a more severe violation results in a more severe penalty), this is called a soft constraint. (Solmaz Kia, No date)

Selection

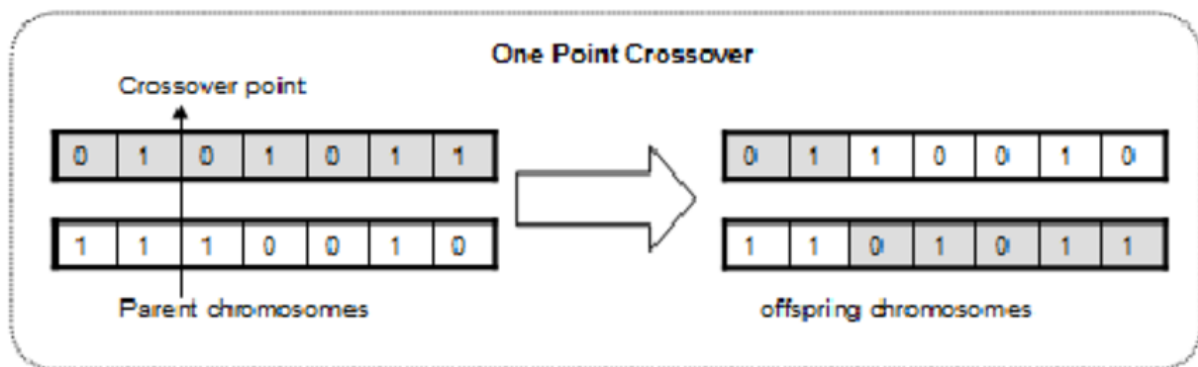
After each generation has been assigned a fitness value by the objective function, the next process is to select the "parents" whose genes will be used to create the next generation of

solutions (the offspring). Typically, the selection algorithms favour the individuals with the most optimal fitness and are selected to produce the offspring. (Manning, et al., 2012).

Gene crossover

In the crossover stage of the genetic algorithm, the genes of two parents (selected using the specific selection algorithm) from the current generation are crossed over to create a set of offspring. An example of a type of gene crossover is one-point crossover (see Figure 3), where a crossover point on the string of parents is selected, and all genes after this point are swapped and this creates 2 offspring. (Dutta, 2019). However, in some optimization problems, not all assortments of chromosomes represent a valid solution (such as in the TSP, which needs one of each city in the solution, and not two of the same city) so many specialised crossover methods have been developed for similar problems. One of the parameters of the genetic algorithm is the crossover rate, which is the probability that two chromosomes cross genes in order to make the new population, or whether the population is entirely made up of certain chromosomes from the previous generation (0% crossover rate) (Chehour, et al., 2016).

Figure 3: **Diagram showing one point crossover** (Kaya, 2011)



Mutation

After a new generation has been created through selection and crossover, mutation occurs. Mutation refers to the random modification of a chromosome and the probability of this occurring is controlled by the mutation rate (Manning, et al., 2012). Mutation is closely

related to the exploration of the search space and allows the solutions to randomly 'jump' across it. It also can prevent the algorithm from getting trapped in local minima as mutation can cause a solution to 'jump' out of it and potentially find a fitter location. Its main role is to maintain genetic diversity in the population. (Abdoun, et al., Not dated)

Additional Parameter: Elitism

Elitism allows the best performing solution(s) in a generation to proceed into the proceeding generation unmodified (Manning, et al., 2012). Elitism is controlled by the elitism rate, similar to the mutation rate and generally helps the algorithm converge faster and removes some stochasticity from the process. However, this can also mean that the algorithm is more likely to prematurely converge at a local minimum and provide a very sub-optimal solution. A balance between mutation and elitism must be held in order to ensure that the search space is being both explored and exploited.

Particle Swarm Optimization Algorithms

Particle swarm optimization (PSO) algorithm is a stochastic population-based metaheuristic that utilises the behaviour of swarm intelligence. Swarm intelligence is the area that deals with many individuals (called particles in this algorithm) and focuses on decentralised control and self-organisation (Dorgio & Birattari, 2007). The PSO algorithm mimics the swarm behaviour of bird flocks hunting for food in a cooperative way. It is based off of the idea of emulating the successes of other particles in the population, and the movement of a particle is influenced by the data of its neighbours and of the swarm (Engelbrecht, 2007). The difference between a neighbourhood and the swarm will be touched on later in this section.

A PSO algorithm contains a swarm of particles, with swarm used as the term for the population, and particle used as the term for the individual solutions. The particles move around in the search space through the addition of a velocity vector to the current position. The velocity vector is determined through the knowledge of the particle and its distance from its best-known position (personal best), and also through the exchange of information from the other particles in the swarm. The experiential knowledge of the individual particle is

known as the cognitive component and the knowledge spread through the swarm (or neighbourhood) is known as the social component. (Engelbrecht, 2007)

There are two types of PSO algorithms: Global Best PSO and Local Best PSO

In the **global best PSO**, (gbest), the neighbourhood for each particle is simply the whole swarm. The velocity is calculated using the following equation,

$$v_{ij}(t + 1) = wv_{ij}(t) + c_1r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]$$

(Engelbrecht, 2007)

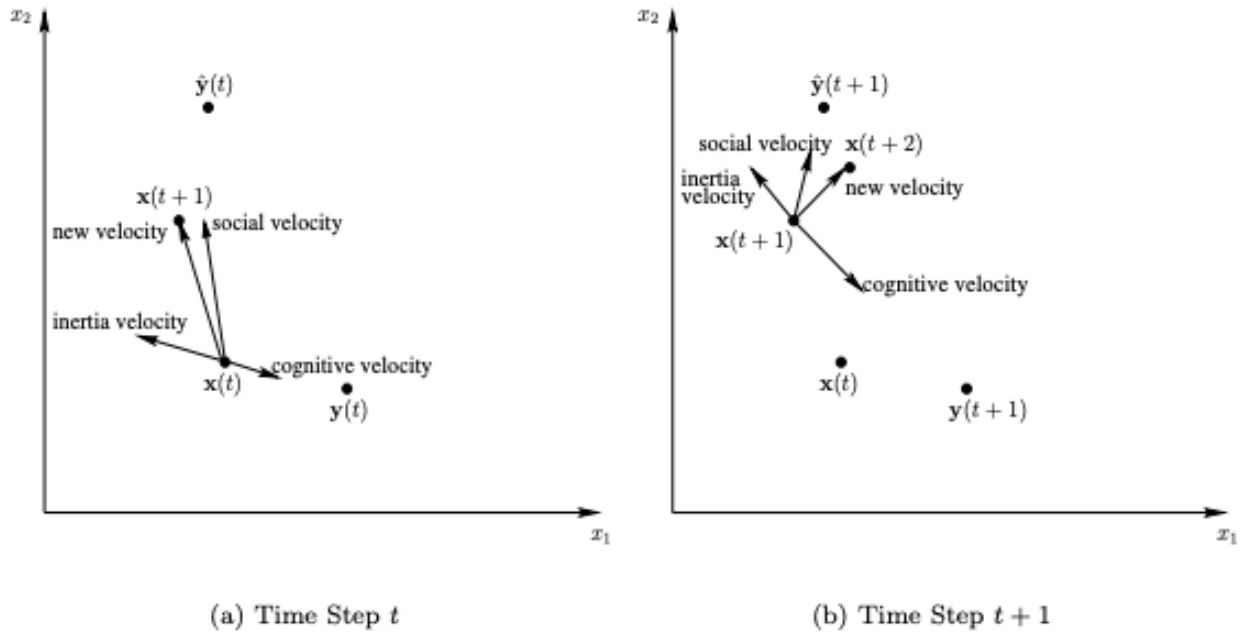
Where:

Table 1: **PSO Equation values and their meanings**

$v_{ij}(t)$	is the velocity of the particle i in dimension j at time step t . $v_{ij}(t + 1)$ is simply the velocity at the next time step (iteration)
$x_{ij}(t)$	is the location of the particle i in dimension j at time step t
w	is the inertia weight that controls momentum. It weighs the contribution of the previous velocity and dictates how it will affect the new velocity
c_1 and c_2	are acceleration constants that are used to scale the contribution of both the cognitive (c_1) and social components (c_2), both are between 0 and 1
$r_{1j}(t)$ and $r_{2j}(t)$	are random values between 0 and 1. This is what gives the algorithm a stochastic element.
$y_{ij}(t)$	is the personal best of particle i since the first time step t
$\hat{y}_j(t)$	is the global best position discovered by any particle at the time step t

(Engelbrecht, 2007)

Figure 4: **Graphical representation of the velocity for a single particle in a two-dimensional search space for two different time steps** (Engelbrecht, 2007)



The c_1 and c_2 components control the stochastic influence of the cognitive and social components on the overall velocity of the particle. c_1 is the cognitive component and refers to how attracted a given particle is to the particle's personal best, c_2 refers to how attracted the particles are to the global best found by the swarm. Particles are most effective when c_1 and c_2 have a good balance and are similar. A c_1 value much greater than c_2 results in excessive wandering, c_1 much less than c_2 can result in premature convergence. (Pyswarms, 2019)

The **local best PSO**, (lbest) is a method where smaller neighbourhoods are created in a ring-like structure in the search space. The social component of the algorithm is within the neighbourhood, instead of within the entire swarm. The vector equation remains largely the same as in gbest except $\hat{y}_j(t)$ is instead $\hat{y}_{ij}(t)$ and is the best position of the neighbourhood of particle i in j dimension instead of the best position of the entire swarm.

The locations of the initial particles can be either randomly generated or initialised evenly across the search space. The initial velocity is set to zero, therefore the velocity of the first iteration of particles will be largely based on the global best of the initial swarm. The initial personal best is assigned to each particle's initial position. The fitness function in PSOAs functions exactly the same as in genetic algorithms.

When to terminate the algorithm can be an important part of the process and choosing the right termination condition for the situation is necessary. The three main stopping conditions are: terminating the program when a pre-set number of iterations has been reached, terminating when a solution is within a certain threshold of the optimum solution, or terminating when there is no improvement within a certain number of iterations, and these are generally applicable to all metaheuristics. (Engelbrecht, 2007)

Hypothesis & Applied Theory

The theory for both metaheuristics has been explained in detail, now it is important to undergo an experiment to test which of the two algorithms is faster and more accurate in solving various optimization algorithms. A qualitative experiment will be carried out where both algorithms will be tested for speed and accuracy against a variety of different optimization problems that belong to different categories and have different characteristics.

I hypothesize that the genetic algorithm will provide a higher accuracy than the particle swarm optimization algorithm, especially in problems with multiple local maxima. This is because genetic algorithm has more of a stochastic element via the mutation rate and has more capability to escape these local minima. I however hypothesise that the PSO algorithm will be faster than the genetic algorithm, this is because trajectory based method used allows for a quicker exploration of the search space. Whereas the genetic algorithm relies on the fact that the new solutions created by crossover and mutation will be better than the previous, and this process is significantly slower than using trajectories and vectors.

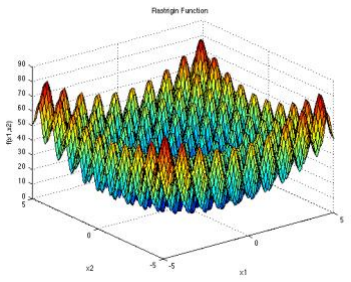
METHOD

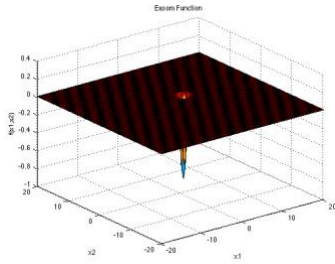
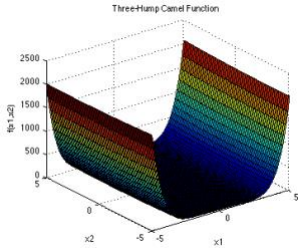
Independent variables

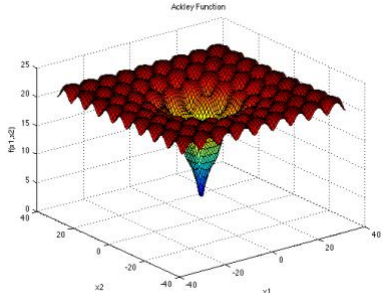
This experiment will be a qualitative investigation rather than a quantitative one. The independent variable refers to what will be changed in the experiment, and I will be changing the characteristics and categories of optimization problems that will be tested on both the

particle swarm optimization algorithm and the genetic algorithm. A test set will be created which includes a collection of optimization problems to be tested. Due to the nature of particle swarm optimization algorithms being trajectory based and only working in a continuous search space, only continuous optimization problems will be included in the test set. In this test set there will be different categories of optimization problems with different characteristics such as those that have many local minima or few local minima, many dimension or 2 dimensions, those that are valley-shaped or have a sharp drop. Only non-linear optimization problems are being used as the simplicity of these linear problems does not lend well to a fair test and often results in luck winning based on previous experimentation. All problems in the test set will have known solutions, this allows for a test for accuracy, as if there is no known solution to a problem, there would be no criterion for testing accuracy.

Table 2: **Test-set of continuous optimization problems. Sourced from** (Surjanovic & Bingham, 2013)

Name	Formula	Global Minimum	Search Domain	Characteristics of function
Rastrigin Function	$f(\mathbf{x}) = 10d + \sum_{i=1}^d [x_i^2 - 10 \cos(2\pi x_i)]$ <p>$d = \text{dimensions}$</p>	$f(0, \dots, 0)$ $= 0$	$x_i \in [-5.12, 5.12]$ for all $i = 1, \dots, d$	<ul style="list-style-type: none"> • Many Local Minima • Dimensions tested: 4 (works with any amount of dimensions) • Non-linear <p>Shown in a 2 dimensional form:</p> 

<p>Easom Function</p>	$f(\mathbf{x}) = -\cos(x_1) \cos(x_2) \exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$	$f(\pi, \pi) = -1$	$x_i \in [-100, 100]$, for all $i = 1, 2$.	<ul style="list-style-type: none"> • Many Local Minima • The global minimum curve has small area relative to the search space • Dimensions tested: 2 (function only works with 2 dimensions) • Non-linear • Flat plain shape with steep drop <p>Shown in a 2 dimensional form:</p> 
<p>Three-Hump Camel Function</p>	$f(\mathbf{x}) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$	$f(0, 0) = 0$	$x_i \in [-5, 5]$, for all $i = 1, 2$.	<ul style="list-style-type: none"> • Valley shape • Three local minima • Dimensions tested: 2 (function only works with 2 dimensions) • Non-linear 

<p>Ackley Function</p>	$f(\mathbf{x}) = -a \exp\left(-b \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2}\right) - \exp\left(\frac{1}{d} \sum_{i=1}^d \cos(cx_i)\right) + a + \exp(1)$ <p>$d = \text{dimensions}$</p> <p>$a = 20$</p> <p>$b = 0.2$</p> <p>$c = 2\pi$</p>	<p>$f(0, \dots, 0)$ $= 0$</p>	<p>$x_i \in [-32.768, 32.768]$, for all $i = 1, \dots, d$.</p>	<ul style="list-style-type: none"> • Many local optima • Non-linear • Dimensions tested: 6 (works with any amount of dimensions) • Flat plain shape with steep drop 
----------------------------	---	---	--	---

Dependent variables

As we are testing for both speed and accuracy of both algorithms as stated in the research question, there will be 2 tests performed on each optimization problem and therefore 2 dependent variables. The first test will be measuring the efficiency of the algorithm using a **fixed-target method** which is the required iterations (in genetic algorithms this refers to each generation and in PSO algorithms this refers to each movement of all particles) to find a solution at a pre-set accuracy target, which will be 0.5 above the global minimum cost value for my experiment. For example, if the optimization problem has a global minimum with a cost value of -1, and the pre-set accuracy target set to 0.5, the number of iterations will be recorded when the cost value reaches $-1 + 0.5$ (-0.5) or closer (Beiranvand, et al., 2017). The number 0.5 was chosen as it requires the algorithm to come somewhat close to the minimum but will not require large amounts of iterations, as we are measuring how fast the algorithm can reach this target, not how accurate this target is. However, for the three-camel hump target of 0.5 was far too easy for the algorithm so the target was changed to 0.00005. This change doesn't affect the experiment as the comparison is between the two algorithms, and since this target was the same for both algorithms it makes no difference to the investigation. The second test will measure the accuracy of the algorithm using a **fixed-cost method**. The

fixed-cost method involves calculating the fixed optimization error by finding the difference between the final best fitness value and the known solution after running the algorithm for a fixed period of iterations. For example, if after 100 iterations of an algorithm minimizing a given fitness function, the final best current fitness is 1.46 and the known best fitness value is 1, then the final optimization error is $1.46 - 1 = 0.46$. The fixed period of iterations selected is 100 iterations, this usually gives enough time for the algorithm to come near the global minimum, but also is a key period where we can observe the algorithms if it gets stuck at a local minimum. It also allows for a quick measurement of results, especially on computers with poor specifications. For each of the two tests performed on each optimization problem, there will be 5 trials, meaning that for each optimization problem, 10 total tests will be performed.

Controlled variables

Table 3: Table of controlled variables

Variable	Description	Specifications
Computer and Operating System	The algorithms will be run on a 13-inch Early 2015 MacBook Air on macOS Catalina 10.15.3	Processor: 1.6 GHz Dual-Core Intel Core i5 Memory: 8GB 1600 MHz DDR3 All other applications will be closed to free up RAM.
Integrated Development Environment (IDE)	The IntelliJ IDEA IDE 2020.3 (Community Edition) will be used to run both algorithms	
Size of population	As both particle swarm and genetic algorithms are population based metaheuristics, the population parameter will be set to 100 for both algorithms. This	

	<p>number was chosen as it allows the algorithms to use their information sharing abilities however is not too computationally taxing.</p>	
<p>Specific parameters for both algorithms</p>	<p>Due to the fact that the two algorithms navigate the search space very differently, there are parameters specific to either algorithm. In genetic algorithms there is the elitism rate, crossover method, and mutation rate. In PSO algorithms there is the c_1 and c_2 components and inertia, w. Finding the optimal parameter configuration for each problem is out of the scope of this investigation, instead default parameters will be used across all problems for both algorithms. These default values are determined by the python libraries for the respective algorithms as there is no agreed-upon expert recommended default values for either algorithm.</p>	<p>Genetic Algorithms:</p> <p>Elitism Rate: 0.01 (1%) Crossover Rate: 0.5 (50%) Mutation Rate: 0.1 (10%) Parents Portion*: 0.3 (30%)</p> <p>PSO Algorithm:</p> <p>$c_1 = 0.5$ $c_2 = 0.3$ $w = 0.9$</p> <p>*Parents portion refers to the portion of the population filled by members of the previous generation. Slightly different to elitism as elitism skips the selection step.</p>
<p>Search space for each optimization problem</p>	<p>Both algorithms will always have the same search space for the same problem, but the search</p>	

	space will change depending on the optimization problems itself.	
--	--	--

Procedure

1. Set up the two algorithms from the code as seen in Appendix A (page 28) in an IntelliJ project folder. The genetic algorithm code is sourced from (Solgi, 2020). The PSO algorithm code is sourced from (PySwarms, 2017).
2. Choose the next optimization problem from the test set and configure it into both programs. See Appendix B (page 30) for examples.
3. Configure the algorithm for the fixed-target method. Set the number of iterations to 1 000 and once the algorithm is done running, the code as seen at the bottom of Appendix A1 and A2 will output how many iterations it took for the fitness function to reach a value of 0.5 or less. Record this number of iterations. Repeat this process for all 5 trials.
4. Configure the algorithm for the fixed-cost method. Set the number of iterations to 100 and record the global best fitness value outputted after termination. Repeat this process for all 5 trials
5. Repeat steps 2-4, changing the optimization problem from the test set.
6. Take averages of the trials

DATA PROCESSING

Below shows the averages for all the problems tested, for raw results refer to Appendix C (page 31)

Table 4: Averaged data for the results for the Rastrigin Function

<u>Rastrigin Function</u>	Average number of iterations to reach	Average optimization error (Difference between final global fitness value

	accuracy target - Fixed-target method	best and known solution fitness value) – Fixed-cost method
Genetic Algorithm	73.4	0.818779715
Particle Swarm Optimization Algorithm	143	1.037905522

Table 5: Averaged data for the results for the Easom Function

<u>Easom Function</u>	Average number of iterations to reach accuracy target - Fixed-target method	Average optimization error (Difference between final global fitness value best and known solution fitness value) – Fixed-cost method
Genetic Algorithm	92	0.1380782328889
Particle Swarm Optimization Algorithm	9.4	0.0000019911266

Table 6: Averaged data for the results for the Three-hump Camel Function

<u>Three-hump Camel Function</u>	Average number of iterations to reach accuracy target - Fixed-target method	Average optimization error (Difference between final global fitness value best and known solution fitness value) – Fixed-cost method
Genetic Algorithm	148.8	0.00110863667

Particle Swarm Optimization Algorithm	22.2	0.00000007786
--	------	---------------

Table 7: Averaged data for the results for the Ackley Function

<u>Ackley Function</u>	Average number of iterations to reach accuracy target - Fixed-target method	Average optimization error (Difference between final global fitness value best and known solution fitness value) – Fixed-cost method
Genetic Algorithm	588.8	16.515100764
Particle Swarm Optimization Algorithm	54.8	0.016053783

ANALYSIS

My first hypothesis that the genetic algorithm will provide a higher accuracy score than the swarm optimization algorithm, was shown not to be true for 3 out of the 4 optimization problems. Through analysing the data seen in the Ackley and Easom functions, it is likely that the genetic algorithm struggles with functions that feature a relatively flat plain with only a small area for the global minimum curve. This can be observed in Figures 5 and 6.

Figure 5: **Graph comparing the iterations against the value of the objective function of the Ackley function for trial 4 of the fixed-cost method for the genetic algorithm**

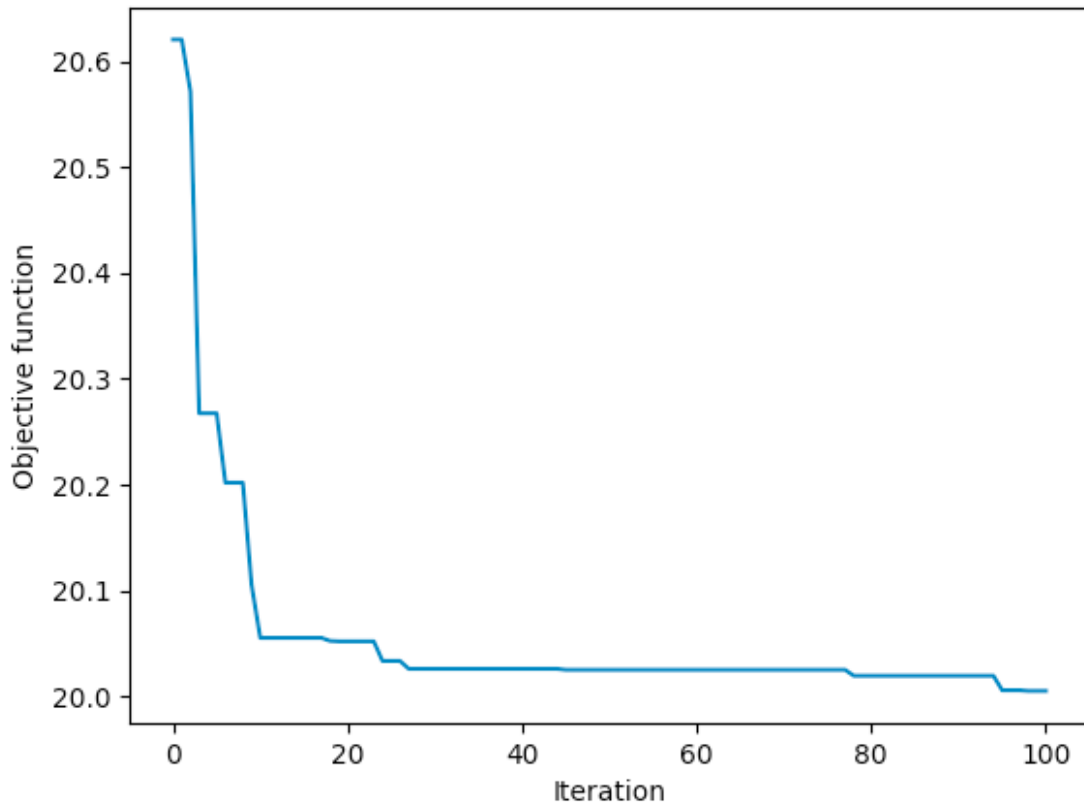
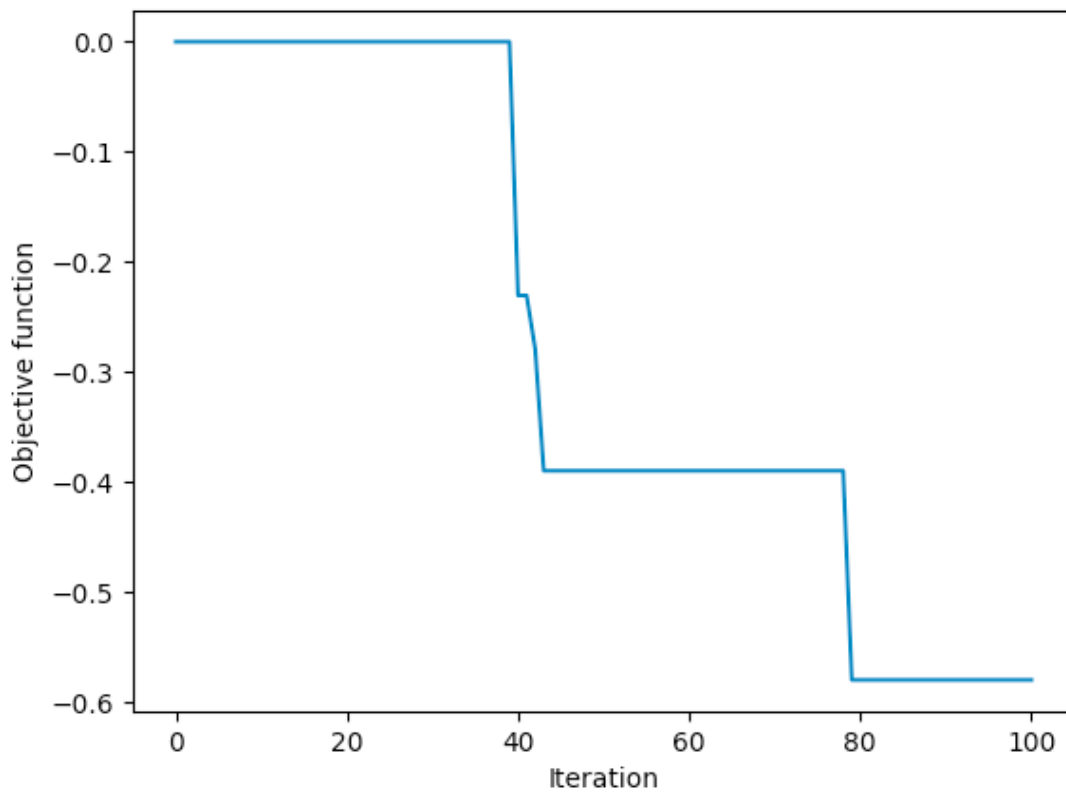


Figure 6: Graph comparing the iterations against the value of the objective function of the Easom function for trial 1 of the fixed-cost method for the genetic algorithm

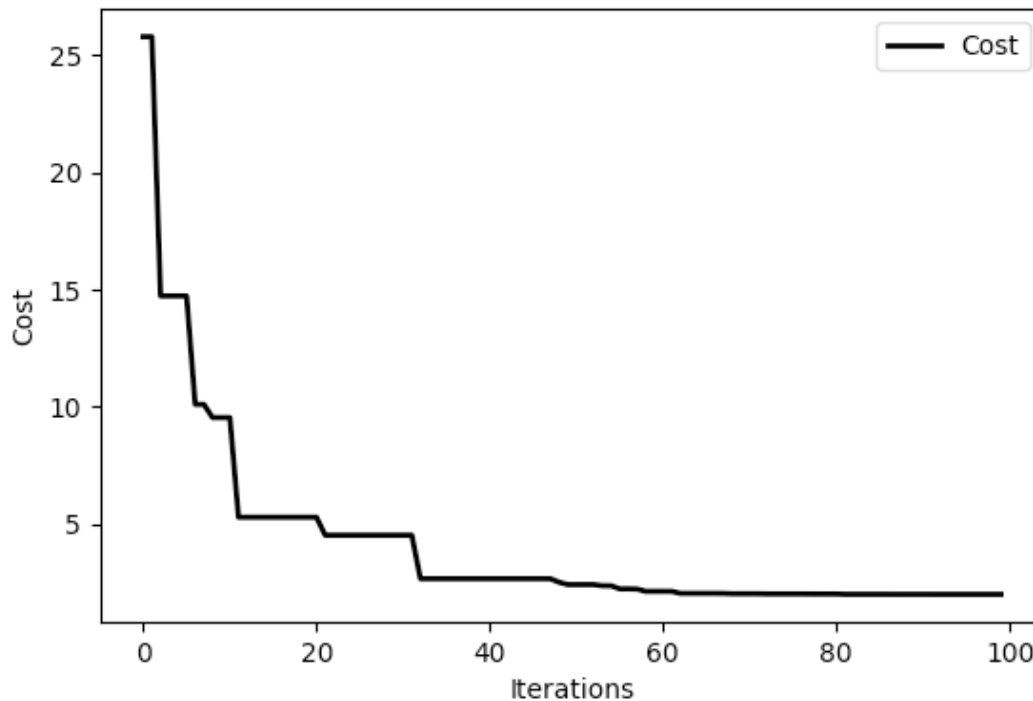


In figure 5, we can observe that the genetic algorithm plateaus around a fitness value of 20 in the Ackley problem, where the global fitness value is 0. This potentially suggests that the genetic algorithm struggles greatly with escaping the local minima in the flat plain of the search space. This issue may partly be due to the high amount of dimensions, however the genetic algorithm surprisingly showed strong results in the Rastrigin function, which features 4 dimensions. A similar scenario is seen in figure 6, with the genetic algorithm getting stuck at multiple local minima of the Easom problem at fitness values of around 0, -0.4, and -0.6, with the global minima fitness value at -1. As the problem has only 2 dimensions, there is less difficulty than in the Ackley problem, however the problem of being unable to escape these local minima arises again. This suggests that the first part of the hypothesis is wrong and that the PSO algorithm yields better accuracy results than genetic algorithms, especially in problems with high amounts of local minima such as the Ackley and Easom problem. The Rastrigin function provided interesting results for the genetic algorithm with it outperforming the PSO algorithm in both accuracy and speed. This may be due to the small search space of the Rastrigin function ($x_i \in [-5.12, 5.12]$) compared to the search space of the Easom problem ($x_i \in [-100, 100]$). The small search space makes it more likely for the stochastic approach of the genetic algorithm to be successful, whereas in a large search space the approach will yield little results with the random jumps across the search space leading to another local minima instead of the global minima curve.

The second part of my hypothesis that the PSO algorithm will be faster than the genetic algorithm can be observed to be correct in 3 out of the 4 problem sets. With the PSO being on average 9.78 times faster than the genetic algorithm in reaching the accuracy target on the Easom function. This is likely due to the fact that within an iteration, the entire population moves drastically whereas the genetic algorithm makes small progress over time, relying mostly on the effectiveness of crossover and random mutations to produce better solutions. Due to the trajectory based method of the PSO algorithm, when a particle is moving towards the gbest which may be stuck at a local minimum, the particle may stumble across the global minimum. This is not the case with genetic algorithms as it often ends up discarding unfit solutions and not selecting them for crossover. It is however important to note that the PSO

algorithm is not immune to getting stuck at local minima and was observed in the Rastrigin function as seen in figure 7.

Figure 7: **Graph comparing the iterations against the value of the objective function of the Rastrigin function for trial 1 of the fixed-cost method for the particle swarm optimization algorithm**



The algorithm plateaus at the end with a final fitness value of only 1.99, with the global optimum fitness value of 0. The Rastrigin function features an increasing amount of local minima as the algorithm approaches the global minimum. The algorithm gets stuck at one of these local minima in 3 out of the 5 trials and it highlights that the PSO algorithm is not perfect either and may need more of a stochastic element such as mutation that could bring the particles closer to the global minimum.

CONCLUSION

This experiment aimed to use theory explained in this essay to design an experiment and practically apply optimization problems onto both the genetic algorithm and the PSO algorithm. The experiment tested for the accuracy and efficiency of the two algorithms

against 4 different continuous optimization problems. The results demonstrated that the particle swarm optimization algorithm almost always performed better on both speed and accuracy. To take it further, the exploration that delved into the theory that caused the patterns behind the results of the experiment and how and why the trajectory based method of the PSO algorithm, while not perfect, outperformed against the mechanisms of selection, mutation, and crossover of genetic algorithms. A limitation of this exploration was the impracticability to fine tune each parameter of the algorithms to perfectly suit each optimization problem in order to gain the most out of the capabilities of each algorithm. This could be an area for further exploration of this investigation.

To answer my initial research question, the particle swarm optimization algorithm vastly outperforms the genetic algorithm in solving optimization problems in terms of both speed and accuracy.

BIBLIOGRAPHY

- Özkaraca, O., 2018. *A review on usage of optimization methods in geothermal power generation*. [Online]
Available at: https://link.springer.com/chapter/10.1007/978-3-030-48453-8_2
[Accessed 2022].
- Abdoun, O., Abouchabaka, J. & Tajani, C., Not dated. *Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem*. [Online]
Available at: <https://arxiv.org/pdf/1203.3099.pdf>
[Accessed 2022].
- Beiranvand, V., Hare, W. L. & Lucet, Y., 2017. *Best Practices for Comparing Optimization Algorithms*. [Online]
Available at:
https://www.researchgate.net/publication/317724973_Best_Practices_for_Comparing_Optimization_Algorithms
[Accessed 2022].
- Blickle, T. & Thiele, L., 1995. *A Comparison of Selection Schemes used in Genetic Algorithms*. [Online]
Available at: <https://tik-old.ee.ethz.ch/file/6c0e384dceb283cd4301339a895b72b8/TIK-Report11.pdf>
[Accessed 2022].

Brownlee, J., 2021. *Local Optimization Versus Global Optimization*. [Online]
 Available at: <https://machinelearningmastery.com/local-optimization-versus-global-optimization/#:~:text=Local%20optimization%20involves%20finding%20the,problems%20that%20contain%20local%20optima.>
 [Accessed 2022].

Car, J., 2014. *An Introducton to Genetic Algorithms*. [Online]
 Available at:
<https://www.whitman.edu/documents/academics/mathematics/2014/carrjk.pdf>
 [Accessed 2022].

Chase, C., Chen, H., Neoh, A. & Wilder-Smith, M., Not dated. *An Evaluation of the Traveling Salesman Problem*. [Online]
 Available at:
[https://scholarworks.calstate.edu/downloads/xg94hr81q#:~:text=The%20brute%20force%20algorithm%20has,complexity%20is%20O\(n!\)%20.](https://scholarworks.calstate.edu/downloads/xg94hr81q#:~:text=The%20brute%20force%20algorithm%20has,complexity%20is%20O(n!)%20.)
 [Accessed 2022].

Chehour, A., Younes, R., Perron, J. & Illinca, A., 2016. *A Constraint-Handling Technique for Genetic Algorithms using a Violation Factor*. [Online]
 Available at: <https://arxiv.org/pdf/1610.00976.pdf>
 [Accessed 2022].

Dorgio, M. & Birattari, M., 2007. *Swarm Intelligence*. [Online]
 Available at:
http://www.scholarpedia.org/article/Swarm_intelligence#Particle_Swarm_Optimization
 [Accessed 2022].

Dutta, A., 2019. *Crossover in Genetic Algorithm*. [Online]
 Available at: [https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/#:~:text=Uniform%20Crossover%20%3A%20Each%20gene%20\(bit,or%20as%20good%20a%20solution.](https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/#:~:text=Uniform%20Crossover%20%3A%20Each%20gene%20(bit,or%20as%20good%20a%20solution.)
 [Accessed 2022].

Dutta, A., 2021. *Encoding Methods in Genetic Algoithm*. [Online]
 Available at: <https://www.geeksforgeeks.org/encoding-methods-in-genetic-algorithm/>
 [Accessed 2022].

Engelbrecht, A. P., 2007. *Computational Intelligence*. [Online]
 Available at: <http://www.shahed.ac.ir/stabaii/Files/CompIntelligenceBook.pdf>
 [Accessed 2022].

FrancQ, P., 2011. *Optimization Problems*. [Online]
 Available at: http://www.otlet-institute.org/wikics/Optimization_Problems.html
 [Accessed 2022].

Gad, A., 2018. *Introducton to Optimization with Genetic Algorithm*. [Online]
 Available at: <https://towardsdatascience.com/introduction-to-optimization-with-genetic-algorithm-2f5001d9964b>
 [Accessed 2022].

Glover, F. & Sörensen, K., 2015. *Metaheuristics*. [Online]
 Available at: <http://www.scholarpedia.org/article/Metaheuristics>
 [Accessed 2022].

Great Learning Team, 2022. *Why is Time Complexity Essential and What is Time Complexity?*. [Online]

Available at: <https://www.mygreatlearning.com/blog/why-is-time-complexity-essential/> [Accessed 2022].

Kaya, Y., 2011. *A Novel Crossover Operator for Genetic Algorithms: Ring Crossover*. [Online] Available at: https://www.researchgate.net/figure/Single-point-crossover_fig1_220485962 [Accessed 2022].

Kumar, D. N., 2020. *Evolutionary algorithms, swarm intelligence methods, and their applications in water resources engineering: a state-of-the-art review*. [Online] Available at: https://www.researchgate.net/figure/Taxonomy-of-optimization-methods_fig1_342112667 [Accessed 2022].

Lin, W., Lian, Z., Gu, X. & Jiao, B., 2014. *A Local and Global Search Combined Particle Swarm Optimization Algorithm and Its Convergence Analysis*. [Online] Available at: <https://www.hindawi.com/journals/mpe/2014/905712/> [Accessed 2022].

Lohn, J., Hornby, G. & Linden, D., 2005. *An Evolved Antenna for Deployment on Nasa's Space Technology 5 Mission*. [Online] Available at: https://link.springer.com/chapter/10.1007/0-387-23254-0_18 [Accessed 2022].

Luenberger, D. G. & Ye, Y., 2008. *Linear and Nonlinear Programming*. [Online] Available at: <https://web.stanford.edu/class/msande310/310trialtext.pdf> [Accessed 2022].

Manning, T., Sleator, R. D. & Walsh, P., 2012. *Naturally selecting solutions*. [Online] Available at: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3813526/> [Accessed 2022].

Mechanical Engineering at IIT Madras, No date. *Optimization Methods*. [Online] Available at: <https://mech.iitm.ac.in/nspch52.pdf> [Accessed 2022].

Mitchell, M., 1999. *An Introduction to Genetic Algorithms*. [Online] Available at: <https://www.boente.eti.br/fuzzy/ebook-fuzzy-mitchell.pdf> [Accessed 2022].

Newcastle University, 2022. *Roulette wheel selection*. [Online] Available at: <http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php>

PySwarms, 2017. <https://pyswarms.readthedocs.io/en/latest/>. [Online] Available at: <https://pyswarms.readthedocs.io/en/latest/> [Accessed 2022].

PySwarms, 2019. *PySwarms Documentation Release 1.0.2*. [Online] Available at: <https://readthedocs.org/projects/pyswarms/downloads/pdf/development/> [Accessed 2022].

Solgi, R., 2020. *geneticalgorithm 1.0.2*. [Online] Available at: <https://pypi.org/project/geneticalgorithm/> [Accessed 2022].

Solmaz Kia, No date. *Lecture 14 Penalty Function Method*. [Online] Available at: <http://solmaz.eng.uci.edu/Teaching/MAE206/Lecture14.pdf> [Accessed 2022].

Surjanovic, S. & Bingham, D., 2013. *Optimization Test Problems*. [Online] Available at: <https://www.sfu.ca/~ssurjano/optimization.html> [Accessed 2022].

Wikipedia, 2022. *Constrained Optimization*. [Online]
Available at: https://en.wikipedia.org/wiki/Constrained_optimization#:~:text=In%20mathematical%20optimization%2C%20constrained%20optimization,of%20constraints%20on%20those%20variables.
[Accessed 2022].

Wikipedia, 2022. *Crossover (genetic algorithm)*. [Online]
Available at: [https://en.wikipedia.org/wiki/Crossover_\(genetic_algorithm\)#Crossover_for_ordered_lists](https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)#Crossover_for_ordered_lists)
[Accessed 2022].

Wikipedia, 2022. *Fitness proportionate selection*. [Online]
Available at: https://en.wikipedia.org/wiki/Fitness_proportionate_selection
[Accessed 2022].

Wikipedia, 2022. *Metaheuristic (Single-solution vs. population-based)*. [Online]
Available at: https://en.wikipedia.org/wiki/Metaheuristic#Single-solution_vs._population-based
[Accessed 2022].

Wikipedia, 2022. *Travelling Salesman Problem (Exact Algorithms)*. [Online]
Available at: https://en.wikipedia.org/wiki/Travelling_salesman_problem#Exact_algorithms
[Accessed 2022].

Wikipedia, 2022. *Travelling Salesman Problem (Heuristic and approximation algorithms)*. [Online]
Available at: https://en.wikipedia.org/wiki/Travelling_salesman_problem#Heuristic_and_approximation_algorithms
[Accessed 2022].

APPENDICES

Appendix A: Programs used in the experiment

Appendix A1: Genetic Algorithm Code

```
import math
import numpy as np
from geneticalgorithm import geneticalgorithm as ga

def f(X):

# FITNESS FUNCTION

return
```

```

algorithm_param = {'max_num_iteration': ???, \
                  'population_size':100, \
                  'mutation_probability':0.1, \
                  'elit_ratio': 0.01, \
                  'crossover_probability': 0.5, \
                  'parents_portion': 0.3, \
                  'crossover_type':'uniform', \
                  'max_iteration_without_improv': None}

# VARIABLE BOUNDARIES
varbound=np.array()

model=ga(function=f,dimension=?,variable_type='real', variable_boundaries=varbound,
algorithm_parameters=algorithm_param)

model.run()

# FOR FIXED-TARGET METHOD
for x in range(0, len(model.report)):
    if model.report[x] <= 0.5:
        print(x)
        break

solution=model.output_dict

```

Appendix A2: Particle Swarm Optimization Algorithm Code

```

import matplotlib.pyplot as plt
import numpy as np
import pyswarms as ps
from pyswarms.utils.functions import single_obj as fx
from pyswarms.utils.plotters import plot_cost_history

max_bound = ??
min_bound = ??
bounds = (min_bound, max_bound)
# Initialize swarm
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}

# Call instance of PSO with bounds argument

```

```

optimizer = ps.single.GlobalBestPSO(n_particles=100, dimensions=??, options=options, bounds=bounds)

# Perform optimization
cost, pos = optimizer.optimize(fx.??, iters=??)
plot_cost_history(cost_history=optimizer.cost_history)
plt.show()

# FOR FIXED-TARGET METHOD
for x in range(0, len(optimizer.cost_history)):
    if optimizer.cost_history[x] <= 0.5:
        print(x)
        break

```

Appendix B: Example Code

Appendix B1: Genetic Algorithm code for Ackley Function

```

import math
import numpy as np
from geneticalgorithm import geneticalgorithm as ga

def f(X): # FITNESS FUNCTION

    dim=len(X)

    t1=0
    t2=0
    for i in range (0,dim):
        t1+=X[i]**2
        t2+=math.cos(2*math.pi*X[i])

    OF=20+math.e-20*math.exp((t1/dim)*-0.2)-math.exp(t2/dim)

    return OF

algorithm_param = {'max_num_iteration': 100, \
                   'population_size': 100, \
                   'mutation_probability':0.1, \
                   'elit_ratio': 0.01, \
                   'crossover_probability': 0.5, \
                   'parents_portion': 0.3, \
                   'crossover_type':'uniform', \
                   'max_iteration_without_improv': None}

varbound=np.array([[ -32.768,32.768]]*2) # BOUNDARIES OF THE SEARCH SPACE FOR THE
FUNCTION
model=ga(function=f,dimension=2,variable_type='real', variable_boundaries=varbound,
algorithm_parameters=algorithm_param)
model.run()

# FOR FIXED-TARGET METHOD
for x in range(0, len(model.report)):
    if model.report[x] <= 0.5:
        print(x)

```

```

        break
solution=model.output_dict

```

Appendix B1: PSO Algorithm code for Ackley Function

```

import matplotlib.pyplot as plt
import numpy as np
import pyswarms as ps
from pyswarms.utils.functions import single_obj as fx
from pyswarms.utils.plotters import plot_cost_history

max_bound = 32.768 * np.ones(2)
min_bound = - max_bound
bounds = (min_bound, max_bound) # BOUNDARIES FOR THE SEARCH SPACE
# Initialize swarm
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}

# Call instance of PSO with bounds argument
optimizer = ps.single.GlobalBestPSO(n_particles=100, dimensions=2, options=options,
bounds=bounds)

# Perform optimization
cost, pos = optimizer.optimize(fx.ackley, iters=100) # FITNESS FUNCTION ALREADY A
FEATURE OF PYSWARMS
plot_cost_history(cost_history=optimizer.cost_history)
plt.show()
for x in range(0, len(optimizer.cost_history)):
    if optimizer.cost_history[x] <= 0.5:
        print(x)
        break

```

Appendix C: Raw Data

<u>Rastrigin Function</u>	Average number of iterations to reach accuracy target - Fixed-target method					Average optimization error (Difference between final global fitness value best and known solution fitness value) – Fixed-cost method				
	T1	T2	T3	T4	T5	T1	T2	T3	T4	T5
Genetic Algorithm	98	60	111	55	43	1.54982	0.56706	1.02741	0.44891	0.50069
Particle Swarm Optimization Algorithm	191	76	49	322	77	1.99006	1.98994	0.21406	0.00040	0.99506

<u>Easom</u> <u>Function</u>	Average number of iterations to reach accuracy target - Fixed-target method					Average optimization error (Difference between final global fitness value best and known solution fitness value) – Fixed-cost method				
	T1	T2	T3	T4	T5	T1	T2	T3	T4	T5
Genetic Algorithm	35	106	141	56	122	0.42074	0.06539	0.20113	0.00313	0.000000374
Particle Swarm Optimization Algorithm	11	7	13	11	5	0.00000228	0.00000184	0.00000406	0.00000011	0.00000166

<u>Three-Hump Camel</u> <u>Function</u>	Average number of iterations to reach accuracy target - Fixed-target method					Average optimization error (Difference between final global fitness value best and known solution fitness value) – Fixed-cost method				
	T1	T2	T3	T4	T5	T1	T2	T3	T4	T5
Genetic Algorithm	140	209	42	155	198	0.0000711	0.0005366	0.0037370	0.0003544	0.0008441
Particle Swarm Optimization Algorithm	14	45	21	15	16	0.000000007	0.000000108	0.000000132	0.000000118	0.0000000237

<u>Ackley</u> <u>Function</u>	Average number of iterations to reach accuracy target - Fixed-target method	Average optimization error (Difference between final global fitness value best and known solution fitness value) – Fixed-cost method

	T1	T2	T3	T4	T5	T1	T2	T3	T4	T5
Genetic Algorithm	356	599	700	641	648	20.0098	2.5428	20.0115	20.0053	20.0062
Particle Swarm Optimization Algorithm	53	44	57	60	60	0.019934	0.018503	0.020196	0.011347	0.010289