Licnachan, Lance Oliver C. **PSOGA**

1 Terms and Definitions

Let us first start with some terms that will be used in this document.

1.1 Optimization Algorithm, Objective and Fitness Functions, and Design Variables

An **algorithm** is a process followed in performing calculations or problem-solving. It uses the English prose to describe every step necessary to complete a task. It is similar to following a cooking recipe or crafting instruction to the letter thereby producing a specific outcome. **Optimization** is a mathematical technique for solving the maximum or minimum value of a function or system. In a broader sense, it is a technique used to solve for the optimum (best) solution to a particular problem. It is therefore defined that an **optimization algorithm** is a process followed in finding the best or most efficient solution to a given problem.

In order to solve a problem, we must create a model that embodies the essence of the said problem. In this sense, the model must be crafted in such a way that we can approach it through computable means. Hence, we define the following terms. An **Objective** or **Fitness Function** is the equation that is modeled after the problem such that, satisfying the function will satisfy the problem. The objective function is important because it will determine the computability and complexity of the problem as well as the approach, in this case, the algorithm and its implementation. Optimization problems aim to obtain the minimum and/or maximum of certain properties of an object. The output of the objective function dictates whether or not a specific input is not only a solution but also the most optimal one. We say, it is a fitness function because it measures the capability and efficiency of the input in solving the problem.

Objective functions take in **Design Variables**. We say design variables because these sequence of numbers are being used to test and determine the quality of the output. We manipulate the value of these variables in order to get the optimal solution. Design often involves trial and error hence, our variables are the trial and error factors that are used to determine whether a set of values satisfy the problem or not. Keep in mind, that although we are doing trial-and-error, there is some sort of intelligence present in the method.

1.2 Heuristic and Meta-Heuristic

A **heuristic** is a technique designed for solving a problem when classical methods are too slow or for finding approximate solutions when classical methods fail to find any exact solution. These classical methods are those that use mathematical identities, properties, and theorems to prove, show, derive or systematically find solutions to problems. The objective of a heuristic is to produce a solution within a reasonable amount of time such that the solution is acceptable enough to the implementor. **Metaheuristic** is a high-level procedure to find, generate or select a heuristic that may provide a sufficiently good solution to an

optimization problem. Since we are dealing with optimization, finding the best, fastest way to solve the problem is also another way of optimization.

1.3 Evolutionary Algorithm

An **evolutionary algorithm** is a generic population-based metaheuristic optimization algorithm. It uses mechanisms inspired by biological evolution such as reproduction, mutation, recombination and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions. Evolution of the population then takes place after the repeated application of the above operators.

1.4 Sets and Sequences

A **set** is a collection of well defined object. In this document, we will talk about sets as a collection of numbers. A set is usually denoted by braces ('{' and '}') and capital letters (A,B,C,D,...) (ex. $A = \{1, 2, 3\}$). In a set, the order of enumeration and repetition of numbers do not matter. That is, $A = \{2, 3, 4, 4, 4, 1\}$ is equal to $A = \{1, 2, 3, 4\}$.

A number is considered an **element** (denoted by $\in$) of set if it belongs to the set. Using our example, we say that 1 is an element of A ($1 \in A$) but 4 is not an element of A ($4 \notin A$).

We say that A is a **subset** of B (written as $A \subseteq B$). If all elements of A are elements of B. If $A = \{1, 2, 3\}$ and $B = \{1, 2, 3, 4, 5\}$ then $A \subseteq B$. However if $C = \{1, 2, 3, 6\}$ then $C \nsubseteq B$.

Cardinality of a set is the number of unique appearances elements in a set. Cardinality is denoted by $|A|$. That is, using our example, $|A| = 3$.

A set without elements is called the **null** or **empty** set ($\varnothing$) that is, $\varnothing = \{\}$. Therefore $|\varnothing| = 0$.

A set with infinite elements is called an **infinite set**.

If we have $A = \{1, 2, 3\}$ and $D = \{3, 4, 5, 6\}$, then the **Union** of A and D (written as $A \cup D$) is the set containing all elements of A and D. That is, $E = A \cup D = \{1, 2, 3, 4, 5, 6\}$.

If we have the same sets A and D, then the **intersection** of A and D (written as $A \cap D$) is the set containing all the common elements of A and D. That is, $E = A \cap D = \{3\}$.

A **Venn Diagram** is a visual representation of the relationships of sets.

In mathematics, numbers are grouped in sets and subsets. We first have the smallest subset, the set of **Natural** or **Whole Numbers** ($\mathbb{N}$) which is the set of counting numbers, {0, 1, 2, 3, 4, 5...}. The next subset is the set of **Integers** ($\mathbb{Z}$) which is the set of natural numbers and their negatives {... -4, -3, -2, -1, 0, 1, 2, 3, 4, ...}. Next are the **Rational numbers** ($\mathbb{Q}$) are the ratios of integers, also called fractions, such as $\frac{1}{2}$, $\frac{-10}{56}$ etc. Next are the **Irrational Numbers**, numbers that are not included in the rational number set such as radicals or roots (ex. $\sqrt{5}$) and numbers having infinite non-repeating decimal places such as π. Finally, the set of **Real Numbers** ($\mathbb{R}$) which consists of both rational and irrational numbers. Other than the real numbers, we have the **Imaginary numbers** ($\mathbb{I}$) which are the numbers that have negative squares. These numbers are involved with the number $i = \sqrt{-1}$. The set containing all numbers is called the **Complex Number** ($\mathbb{C}$). This set is the union of both real and imaginary numbers. These numbers are usually represented by the sum of

a real and an imaginary number (ex. $1 + i$). A Venn Diagram of the number sets is given by figure 1 .

A **sequence** is a set of enumerated numbers wherein the oder of enumeration is important.

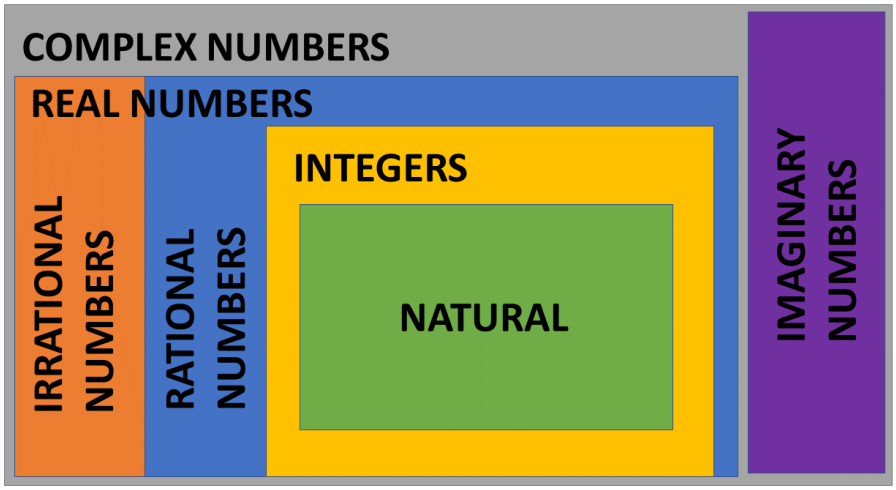

Figure 1: A Venn Diagram Showing the Relationship of Number Sets

1.5 Lines, Planes and Dimensions

A **coordinate** is a sequence of numbers that determine the position of points on objects. This is commonly used in denoting location in space. We usually have a reference from which we determine how far a point or object is located from the reference point. **Dimension** is the number of coordinates needed to specify a point on an object.

A **line** is a straight one-dimensional figure having no thickness. A **Line** is made up of infinite points as it extends infinitely in both directions. A **Real Number Line** is therefore, defined as a line wherein every point represents a real number. This makes sense because the set of real numbers is infinite. Since each point is represents a real number, the coordinate of any point on the line is given by the real number. A visual representation of a real number line is shown on figure 2. As previously stated, if we want to know the coordinates of a point on the line, we simply tell what number the point represents. Hence, we know the distance from which the point is located from our reference point, 0.

A **plane** is a two-dimensional surface. Ruled and spanned by two independent perpendicular lines. A **coordinate plane** is a plane that is spanned by the real number lines, x-axis and y-axis hence, it is also known as the space R^2. Each point on this plane represents a pair of coordinates (x, y). We usually assign the first number, x, for the distance on the x-axis and the second number, y, for the distance on the y-axis. A coordinate plane is shown on figure 3. As we can see, the black point is said to be located at (1,4) this means that it is 1 unit away from (0,0) on the x-axis and 4 units away from (0,0) on the y-axis.

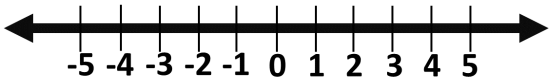

Figure 2: A Real Number Line

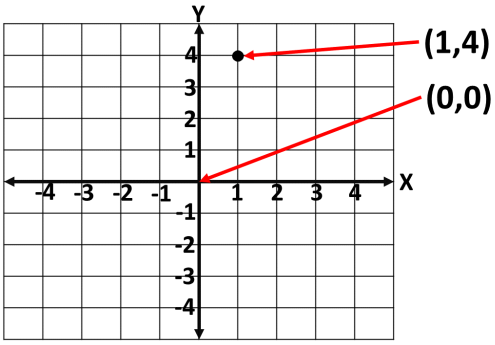

Figure 3: A Coordinate Plane

1.6 Vectors and Arrays

A **Vector** is a quantity having both magnitude and direction. In a coordinate plane, it is represented by an arrow as shown in figure 4. We can see that vector $a =< 1, 1 >$ is 1 unit to the right of the point (0,0) and 1 unit above the point (0,0). Magnitude is visually represented in length. Direction, one the other hand, is visually represented by the arrowhead. It is broken down into two parts, x and y components. X component is its horizontal length while the Y component is its vertical length. The vector's magnitude ($|a|$) is given by the Pythagorean theorem: $|a| = \sqrt{(x^2 + y^2)}$ where x and y are its x and y components.

The negative of a vector is simply a vector having the same magnitude but of opposite direction as seen on figure 4. We can see that the vector $-a$ has the same magnitude but opposite of the direction of vector a. Adding and subtracting vectors are simple in that each component of vector A is added or subtracted to the respective components of vector B. For addition, $C = A + B$ can be written as $(x, y) =< 1, 2 > + < 3, 4 >=< 3, 6 >$. For subtraction, $C = A - B$ can be written as $< x, y >=< 1, 2 > +- < 3, 4 >=< 1, 2 > + < -3, -4 >=< -2, -2 >$. An example is seen on figure 5. As we can see, if we add two vectors, $V_1 and V_2$, the resulting vector $< 4, 3 >$ is longer than both vectors if they are both in the same direction. If we subtract the vectors, $V_1 and V_2$ the resulting direction will depend on which vectors are considered as the minuend and subtrahend.

If we consider a vector in dimension 3, then we will have to add to its components. Its components are now x, y and z where x is its length, y is its height and z is its width. In

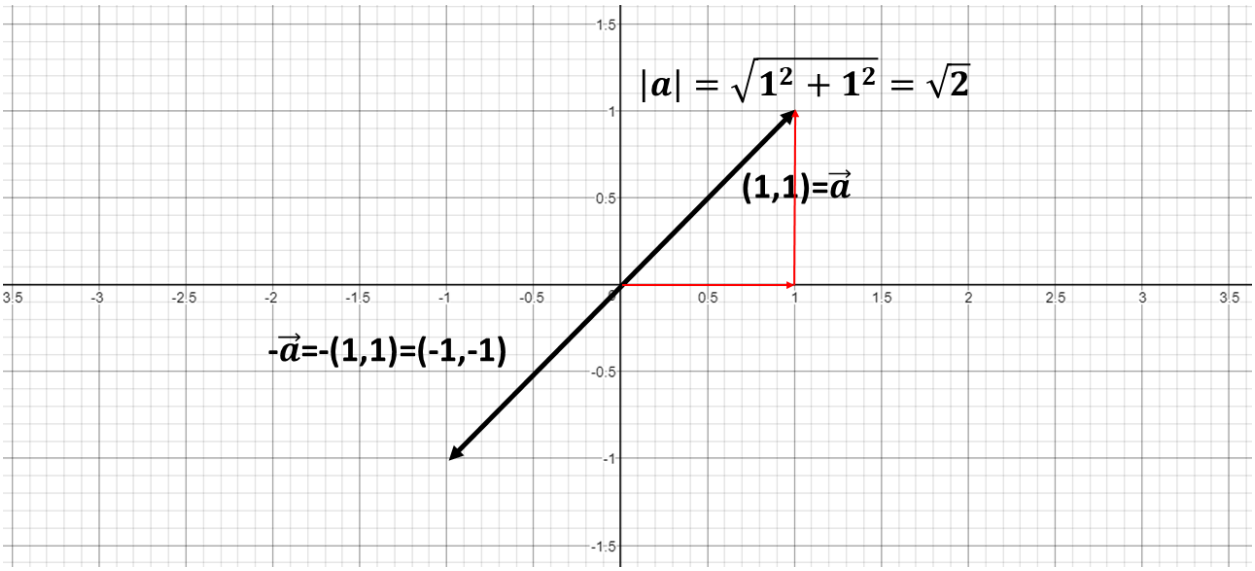

Figure 4: Vectors in a Coordinate Plane

general, if we have a vector in dimension n, it is defined with n components.

In this document, we consider the velocity of an object inside a defined virtual space of dimension n. **Velocity** is defined in physics as speed with direction. For example, if an object has a speed of 9 m/s then we can say that the object is simply covering a distance of 9 metric units at each time step but if we state that the object has a velocity of 9 m/s to the right, then we can say that the object is covering a distance of 9 metric units at each time step to the right of its current position. It is important that take note that vectors usually involve two ordered n-tuples that give its original and final positions.

An **Array** is a collection of objects, having shared some similar properties, arranged in a particular order. An array is usually contained in rows and columns. Arrays are denoted by the syntax ArrayName[Size][Size] wherein every [] denotes a dimension. For example we have the array MyArray[3] it is an array of one-dimension having 3 elements. Take note that the size is sometimes omitted to represent variability. In simple terms, an array is like a series of boxes that contain elements with some similar properties. If we have an array of dimension 2 (MyArray[X][Y]) then we have X rows of Y boxes. A visual representation is shown on figure 6. As we can see, the array A[2][5] has two rows and 5 columns. Each element occupies a single box.

It is common notation to access elements of arrays by its index. Indexing usually starts from 0. In figure 6 the red numbers indicate indices of the elements. For example, if we want to access the first element in A from figure 6, we say A[0][0]. If we want to access element 'H' in A, we say A[1][2].

In this paper we will be dealing with arrays that whose element are vectors and coordinates.

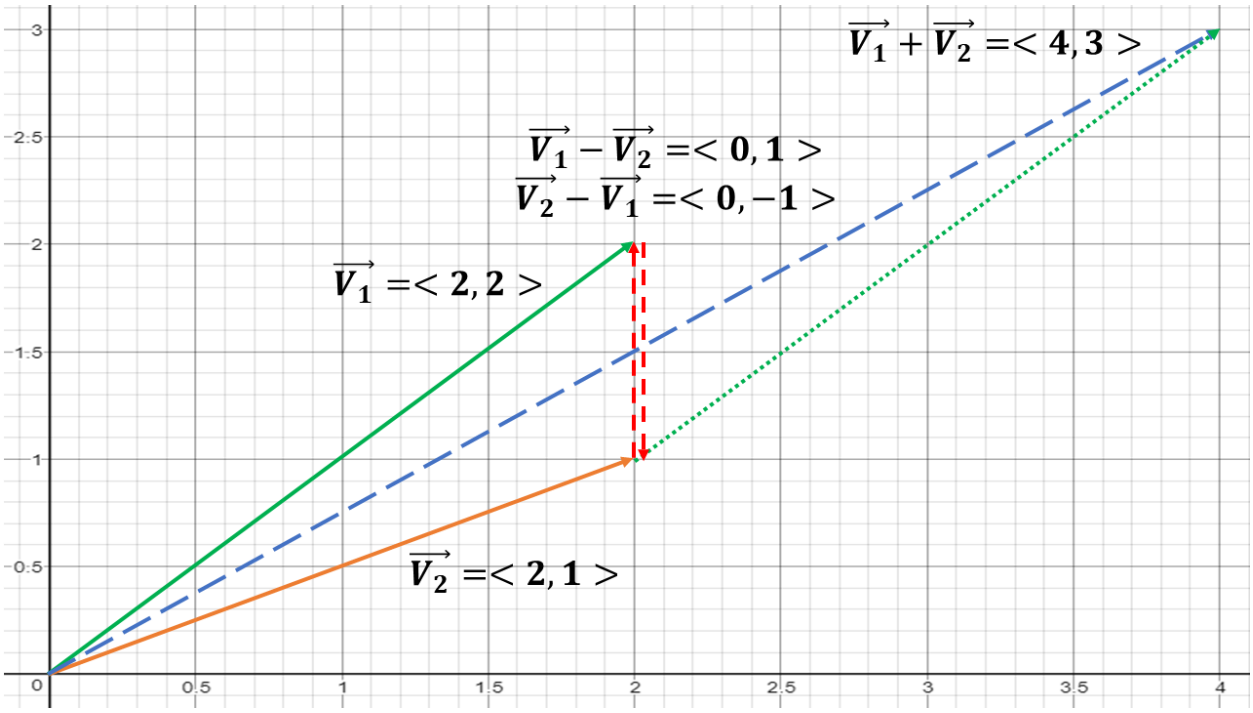

Figure 5: Adding and Subtracting Vectors (Coordinate Plane)

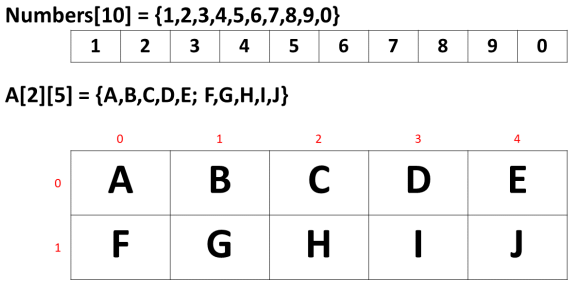

Figure 6: Visual examples of arrays

1.7 Birds, Flocks, Swarm and Particles

Flock is the collective term for a group of birds. **Swarm** is a collective term used in this document to represent a group of particles. **Particles** are mass-less and volume-less points that represent real-world objects and traverse a defined search space. **Search space** is the area or virtual space that delimits the values that can be taken by particles, moreover, it is composed of real numbers that contains a solution to the problem being evaluated.

1.8 Simulations and stochastic variables

A **simulation** is a computational model that imitates real world situations and processes. These usually involved an implementation of mathematical equations that employ stochastic variables for a more 'lifelike' appearance. A **stochastic variable** is a variable whose value is a random number usually taken from a uniform distribution from 0 to 1. That is, every

number between 0 and 1 has an equal chance to be selected.

1.9 Natural Selection and Survival of the Fittest

Natural Selection is the process by which organisms with better attributes adapted to the environment tend to increasingly survive and transmit their genetic characteristics through generations. **'Survival of the fittest'** is a phrase by Charles Darwin that describes the mechanism of natural selection. It is best understood as survival through reproductive multiplicity. That is, the more survivability an individual has, the more it is likely to reproduce, hence it's genes are more likely to be transmitted to the next generation.

In natural selection, there is a variation on traits that is to say that individuals have differences in certain attributes such as height, length, shape etc. It is important to note that not all individuals reproduce to the full potential because the environment has a certain limit to the number of creatures it can sustain. The passing of characteristics or traits from one generation to another is called **heredity**. The more advantageous traits is more commonly passed on and retained because they help the individual or group to survive.

1.10 Natural Genetics, Recombination and Mutation

Gregor Mendel is the father of genetics. He discovered the mechanics of heredity or how traits are being passed down from parents to offspring. During cell division, thread-like structures located inside the nucleus of animal and plant cells called **chromosomes** are replicated. These chromosomes contain the genes which dictate the attributes of the individual. They tell how the body is to be built and how it functions. **Recombination** is the rearrangement of the genetic material by exchanging the same gene subsegments of two chromosomes (one from each parent) which allows for the creation of a new individual that has characteristics similar to those of the father and of the mother. Note that the exchanging process may occur in multiple areas of the strands. **Mutation** on the other hand is the alteration of genes resulting from an error during replication. This results in unique characteristics that may be new from the gene pool of the previous generation. Mutation may be good or bad for the individual but this phenomenon has a low chance of occurring naturally for every generation.

1.11 Robustness

Robustness is the balance between efficiency and efficacy necessary for the survival in many different environments. For Algorithms, this translates to consistent efficiency under different problems areas such that there is little to no change in the process. This means that there is less cost for redesigns. Note that nature is the best example in terms of robustness. It tries to maintain and cope with the many different changes that occur everyday. Hence, we have evolutionary algorithms as stated above.

1) Initialize a population (array) of particles with random positions and velocities on d dimensions in the problem space.
2) For each particle, evaluate the desired optimization fitness function in d variables.
3) Compare particle's fitness evaluation with particle's *pbest*. If current value is better than *pbest*, then set *pbest* value equal to the current value, and the *pbest* location equal to the current location in d-dimensional space.
4) Compare fitness evaluation with the population's overall previous best. If current value is better than *gbest*, then reset *gbest* to the current particle's array index and value.
5) Change the velocity and position of the particle according to equations (1) and (2), respectively:

$$v_{id} = v_{id} + c_1 * rand( ) * (p_{id} - x_{id}) + c_2 * Rand( ) * (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

6) Loop to step 2) until a criterion is met, usually a sufficiently good fitness or a maximum number of iterations (generations).

Figure 7: PSO Algorithm[3]

1.12 Exploration and Exploitation

Exploration is the capability of the algorithm to search solutions in parts of the subspace it has not yet taken into consideration. **Exploitation** is the capability of the algorithm to utilize known data in searching for solutions in the search space.

2 PSO

Particle Swarm Optimization (PSO) is an optimization algorithm based on a simplified social model. PSO was proposed by Kennedy and Eberhart on 1995. [1, 2] The algorithm is shown on figure 7. PSO was discovered from attempts to simulate bird flocking and fish schooling. It has been used to solve a wide array of optimization problems ranging from simple root finding to complex engineering optimization problems.

3 Background

We first discuss the concepts where this algorithm was based upon. Early computer animations used to simulate a flock of birds by individually giving each bird a script to follow, this includes motion, direction, and speed. Each bird was much like an actor in a play, performing actions under a set of instruction. The problem was that it was not scalable. Animators could not possibly give individual scripts to thousands of birds within a short period. This type of approach is too inefficient. This is why, scientists such as Reynolds[4], Heppner and Grenander [5] have tried to simulate movements of birds and fish using the

computational power of computers. They tried to simulate where birds would fly to in every time step or frame in the animation. These simulations were using mathematical and physical concepts to mimic the unpredictable movements of birds when they fly in groups. The initial tests were made such that a population of birds were created, each having its own velocity and initial position on a defined space of definite dimension. These birds were "flying" through the virtual space created by simply adding each bird's velocity to its current position at each time step. Their velocities would change each time step according to the velocities of the nearest neighboring birds to avoid collisions. The initial tests showed that direction and speed were not enough to capture the natural flocking of birds this is because after several time steps, the whole flock would unanimously uniformly fly through the defined space in an unchanging direction. This resulted in the introduction of a craziness factor in the form of stochastic variables multiplied to the velocities of each bird. This change resulted to simulations looking much more lifelike.

Let us take, for example, two birds A and B on a real number Cartesian plane. If bird B is flying at a rate of 9 units per second forward and 5 units per second upward and bird B is bird As neighbor, bird A will change its velocity to match bird Bs velocity. Hence, bird A will have a flying rate of 9 units per second forward and 5 units per second upward with each value multiplied to a random number uniformly distributed from 0 to 1. This means, bird A might not fully replicate the velocity that bird B has. This is seen in nature as bird A trying to approximate the velocity of bird B in such a way that they will not collide.

The next step towards development was the introduction of a focal point to which the flock would move towards. This was introduced as a "roost" by Heppner[5], typically it is a point in space that indicated where the flock would finally land. Upon simulating this, the birds already have a "lifelike" appearance which therefore allowed the elimination of the 'craziness' factor. It was then noted that birds usually land where there is food, hence the roost was replace by a vector called the "cornfield vector" which is a two-dimensional vector of XY coordinates on the Cartesian plane. Given a known position of food, the birds now changed their velocity according to the distance between their current position and the cornfield vector. Each bird now "remembers" the closest position values it was at during that time step. It also took in consideration the closest position values that any bird in the population has been in. Each bird now changed their velocities with the values that they remember.

The algorithm was then extended to spaces with multiple dimensions. The algorithm was tested from the singular dimesion space R, then to the coordinate system R^2 and finally to the 3-dimensional space, R^3. It was generalized that the algorithm would work in any number of dimensions.

The velocity equation underwent some changes until it became:

$$V_x[][] = c1 * rand() * (pbestx[][] - presentx[][]) + c2 * rand() * pbestx[][gbest] - presentx[][]) \quad (1)$$

where $v_x[][]$ is the x velocity component of a particle in n dimensions, $rand()$ is a stochastic variable, $pbestx[][]$ is the x component of the particle's best position in n dimensions, $pbestx[][gbest]$ is the x component of the population's best particle position in n dimensions, $presentx[][]$ is the x component of the particle's current position in n dimensions, $c1$ and $c2$ are constant numbers.

Eberhart and Kennedy [1] adopted the term swarm from Millonas under the circumstance that the behavior of the members of the population satisfies the 5 principles of swarm intelligence as proposed by Millonas. These 5 principles are:

1. proximity principle - members are able to carry out simple space and time calculations

2. quality principle - members respond to the quality factors of the environment

3. principle of diverse response - members do not cp,,ot tp excessively narrow channels

4. principle of stability - members do no change the mode of behavior everytime the environment changes

5. principle of adaptability - members are able to change their mode of behavior when it is worth the computation price

The members of the population satisfy these principles because

1. The population carries out n-dimensional space calculations over a series of time steps

2. Each member responds to the quality of the personal best and global best variables

3. The allocation of responses between personal best and global best ensures diversity of response.

4. The population changes its overall mode of behavior only when the global best changes.

5. The population is adaptive because it does change when the global best changes.

4 Further Developments

Eberhart and Shi[3] explains that the terms of the velocity vector seen on figure 7 are all important. The first term being the previous velocity value gives 'memory' to the particle. It keeps the particle at a good position until a better position is found. Without it, the particle will fly towards the centroid of the pbest and gbest. In addition, without it, the search space will shrink and never grow since it will only move toward the centroid of it's recorded pbest and gbest. The two terms concerning the personal best and global best comparison is necessary to keep the particles from flying in the same direction and leaving the search space.

Eberhart and Shi[3] further improved the original algorithm proposed by Eberhart and Kennedy[1] as seen on figure 7 by introducing inertia weight. This inertia weight is responsible for balancing global and local exploration. The new velocity equation becomes $v_{id} = v_{id} * w_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * rand() * (p_{gd} - x_{id})$ where w_{id} is the inertia weight. Eberhart and Shi states that having a high inertia weight ($w > 1.2$) results in more global exploration but less chances of finding the optima because the particles keep exploring new regions in the space. In contrast, having a low inertia weight ($w < 0.8$) will converge to local optima quickly but will not ensure that the global optimum value will be found. Low inertia weight allows for a fine exploration of a region in the space. Having an inertia weight between 0.8 and 1.2 gives the best chances of finding a global optimum but will take a moderate number of iterations. They theory crafted that it is best to have a high inertia weight in the beginning for extensive global exploration and then reducing the inertia weight gradually through time for a more refined search on local areas. Although the study does give a good background as to the selection of such numbers, in implementing PSO, one must also take in consideration that not all problems are the same hence, implementor must tweak the PSO variables to suit the problems they are trying to solve.

Although PSO is simple in implementation and design, it had certain flaws. It has high computational costs which is given by it slow convergence.[6] Convergence is a problem for PSO because of the restrictions imposed on the velocities of the particle, in addition, although it converges to a point, the particle are ever moving which causes the particles to be in perpetual oscillation around the optima. The population may converge but due to the perpetual motion, convergence can become a problem, if high precision is taken in consideration, the population may not at all converge. Hence, many studies try to solve such problems.

An innovation to the PSO is the introduction of a constriction factor K necessary for ensured convergence introduced by Clerc[7]. The formula then becomes $v_{id} = K[v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * rand() * (p_{gd} - x_{id})]$ where $K = \frac{2}{|2-\varphi-\sqrt{\varphi^2-4}|}$, $\varphi = c_1 + c_2$ and $\varphi > 4$.

PSO has been improved by combining it with other optimization algorithms as well. These hybrids will be explored in the later sections.

5 Example

Consider the equation $F(x) = x^2+x-6$. If we want to find the roots of $F(x)$, where $F(x) = 0$. We can employ PSO to find this. Our fitness function will then be $f(x) = |F(x)| = |x^2+x-6|$. The closer the value of $f(x)$ to 0, the better the x. We also define our search space as $x \in (-10, 10)$.

We first generate a population of 8 members. We have the array Population[8][3] where the first column (Population[n][0], $n \in [0, 7]$) indicates location, second column (Population[n][1], $n \in [0, 7]$) for velocity, and third column (Population[n][2], $n \in [0, 7]$) for current fitness. We also create an array Pbest[8][2] where the first column (Pbest[n][0], $n \in [0, 7]$) for personal best fitness and second column (Pbest[n][1], $n \in [0, 7]$) for personal best position. We also set an array GBest[3] where the first entry (Gbest[0]) contains the row number of the member having the best global position, the second entry (Gbest[1]) contains the member's best fitness and the third entry (Gbest[2]) contains the member's best position. We also have

an array TransPopulation[8][2] that will hold the values of the next locations and velocities of current population in the array Population[8][3]. In TansPopulation, the first column (Population[n][0], $n \in [0, 7]$) indicates next location and second column (Population[n][1], $n \in [0, 7]$) is for next velocity.

5.1 ITERATION 1

We randomly choose numbers from the set of real numbers from -10 to 10 for the location of each member. We also randomly choose numbers from the set of real numbers from -3 to 3 for the velocity of each member. At max, each member can only travel 3 units to the left or 3 units to the right. *Note that in this example, we will be using integers so that it will be easy to compute and visualize. Any real numbers can be used just as long as they are within the specified range given. Suppose we get the population:

Population[8][3]=

1	+2	
7	+2	
7	-1	
-7	0	
-6	+1	
4	+2	
4	-3	
0	+3	

Next in our algorithm is to evaluate each member's fitness under the fitness equation, $f(x)$. For member 1, $x = 1$, we calculate $f(x)$. $f(x) = |(x^2 + x - 6)| = |(1^2 + 1 - 6)| = |-4| = 4$. We do this for the rest of the members and place the answers in the third column of our Population array We get:

Population[8][3]=

1	+2	4
7	+2	43
7	-1	43
-7	0	36
-6	+1	18
4	+2	14
4	-3	14
0	+3	6

Since this is the initial iteration, we set the personal best fitness and location of each member to its current location and fitness. We get:

Pbest[8][2]=

4	1
43	7
43	7
36	-7
18	-6
14	4
14	4
6	0

Since this is the initial iteration, we set the global best fitness and location. We first get the lowest value in the first column of Pbest which is 4. In this case, we have 2 members having the value of 4. We can choose arbitrarily so we choose the first member. We then copy the location and fitness of the member to the Gbest array and set the first element to the row number of the first member. Gbest is therefore:

Gbest[3]=

0	4	1

Next we change the velocity of each member by the equation 1 in our algorithm. We first choose $c1 = c2 = 2$. We need to implement randomly generated numbers from 0 to 1 but since we are simulating, we choose it to be 0.5 for all members. For member 1, $x_0 = 1$ (note that the subscript denotes the row number of each member). We get $v_0 = v_0 + (2)(0.5)(Pbest[0][1] - Population[0][0]) + (2)(0.5)(Gbest[2] - Population[0][0]) = 1 + (1-1) + (1-1) = 1 + 0 + 0 = 1$.
For member 2, $x_1 = 9$ (note that the subscript denotes the row number of each member). We get $v_1 = v_1 + (2)(0.5)(Pbest[1][1] - Population[1][0]) + (2)(0.5)(Gbest[2] - Population[1][0]) = 2 + (7-7) + (1-7) = 2 + 0 + (-6) = -4$. Since we imposed that velocities should only be from [-3,3] we set $v_1 = -3$. This is done to limit the 'flying' range of members in order to make it look 'natural'.
We do this for all members and save them in the second row of our TransPopulation array. We get:

TransPopulation[8][2]=

	+1
	-3
	-3
	+3
	+3
	-1
	-3
	+3

Next we change the current position of each member by the equation 2 in our algorithm. For member 1, $x_0 = 1$ (note that the subscript denotes the row number of each member). We get $x_0 = x_0 + v_0 = Population[0][0] + TransPopulation[0][1] = 1 + 1 = 2$. We do this for all members and save them in our first column of our TransPopulation array. We get:

TransPopulation[8][2]=

2	+1
4	-3
4	-3
-4	+3
-3	+3
3	-1
1	-3
3	+3

Now we talk about the stopping criterion. Usually the stopping criterion is when a certain number of iterations is reached hence, it is said that the test has failed. The other stopping criterion is when all of the members have the same fitness values and the fitness value is equal to the goal or just close to it. Another success oriented stopping criterion is when the fitness value is close to the goal and is the same for n iterations, this n is determined by the user. This 'closeness' is relative with respect to the precision you would consider as a success since most or some problems merely require a very accurate and precise answer. In this example, if we considered real numbers, the algorithm might not exactly find 0 but we can say that it is sufficient enough to get a value with at least an 7 digit precision that is $f(x) \leq 1x10^{-}7$. For this case, our goal is that $f(x)$ is equal to 0 since we are only considering integers, we are sure that 0 will be found. This will be judged as a success.
We choose that our maximum iteration is 10 and if that is reached, the simulation has failed. Since not all members' fitness values are not near the goal, and we have just started, we loop the process.

5.2 ITERATION 2

We replace the values of the first and second column of the Population[8][3] with our TransPopulation[8][2]. We also calculate the fitness values of each new member. For member 1, $x = 2$, we calculate $f(x)$. $f(x) = |(x^2 + x - 6)| = |(2^2 + 2 - 6)| = |0| = 0$. This is good because we reached, that is for $f(x) = 0$.

Population[8][3]=

2	+1	0
4	-3	14
4	-3	14
-4	+3	6
-3	+3	0
3	-1	6
1	-3	4
3	+3	6

We now compare each member's current fitness (Population[n][3], $n \in [0, 7]$) with its recorded personal best fitness (Pbest[n][0]). If its current fitness is better than the recorded personal best, change it. For the first member, $x_0 = 2$, we compare its current fitness and personal best fitness. Population[0][3]¡Pbest[0][0]? that is, 0¡4? yes. Since it is affirmative, we change

the Pbest[0][0] to 0 and Pbest[0][1] to 2. If it was negative, we do not change the values of the member's Pbest. We do this comparison for all other members.

Pbest[8][2]=

0	2
14	4
14	4
6	-4
0	-3
6	3
4	1
6	3

We now get the lowest value under the first column of Pbest. We get 0. Again, we have two members having the value 0. We choose the first element again, remember that this selection is arbitrary. We can choose any of the two since they both have the lowest value. Since 0 is less than our current Gbest[2]=4. We change the values of Gbest.

Gbest[3]=

0	0	2

We now again calculate the new velocities and positions of every member.

TransPopulation[8][2]=

3	+1
-1	-5
-1	-5
-1	+3
0	+3
1	-2
-1	-2
5	+2

Since not all the members are not near the goal, and Iteration 2¡10, we loop the process.

5.3 Iteration 3

We replace the values of the first and second column of the Population[8][3] with our TransPopulation[8][2]. We also calculate the fitness values of each new member. For member 1, $x = 3$, we calculate $f(x)$. $f(x) = |(x^2 + x - 6)| = |(3^2 + 3 - 6)| = |6| = 6$. This is bad because we already reached the goal but now it went past it.

Population[8][3]=

3	+1	6
-1	-5	6
-1	-5	6
-1	+3	6
0	+3	6
1	-2	4
-1	-2	6
5	+2	24

We now compare each member's current fitness (Population[n][3], $n \in [0, 7]$) with its recorded personal best fitness (Pbest[n][0]). If its current fitness is better than the recorded personal best, change it. For the first member, $x_0 = 3$, we compare its current fitness and personal best fitness. Population[0][3]¡Pbest[0][0]? that is, 6¡0? No. Since it is negative, we do not change the values of the member's Pbest. We do this comparison for all other members.

Pbest[8][2]=

0	2
6	-1
6	-1
6	-4
0	-3
4	1
4	1
6	3

We now get the lowest value under the first column of Pbest. We get 0. Since 0 is equal to our current Gbest[2]=0. We do not change the values of Gbest.

Gbest[3]=

0	0	2

We now again calculate the new velocities and positions of every member.

TransPopulation[8][2]=

2	-1
-3	-2
-3	-2
2	+3
2	+2
2	+1
2	+3
2	-3

Since not all of the members' fitness values are not near the goal, and Iteration 3¡10, we loop the process.

5.4 Iteration 4

We replace the values of the first and second column of the Population[8][3] with our TransPopulation[8][2]. We also calculate the fitness values of each new member. For member 1, $x = 2$, we calculate $f(x)$. $f(x) = |(x^2 + x - 6)| = |(2^2 + 2 - 6)| = |0| = 0$. This is good because we reached, that is for $f(x) = 0$.

Population[8][3]=

2	-1	0
-3	-2	0
-3	-2	0
2	+3	0
2	+2	0
2	+1	0
2	+3	0
2	-3	0

We now compare each member's current fitness (Population[n][3], $n \in [0, 7]$) with its recorded personal best fitness (Pbest[n][0]). If its current fitness is better than the recorded personal best, change it. For the first member, $x_0 = 2$, we compare its current fitness and personal best fitness. Population[0][3]¡Pbest[0][0]? that is, 0¡0? No. Since it is negative, we do not change the values of the member's Pbest. We do this comparison for all other members.

Pbest[8][2]=

0	2
0	-3
0	-3
0	2
0	-3
0	2
0	2
0	2

We now get the lowest value under the first column of Pbest. We get 0. Since 0 is equal to our current Gbest[2]=0. We do not change the values of Gbest.

Gbest[3]=

0	0	2

We now again calculate the new velocities and positions of every member.

TransPopulation[8][2]=

1	-1
0	+3
0	+3
5	+3
4	+2
3	+1
5	+3
-1	-3

Since all of the members' fitness values are equal to the goal and we have not reach 10 iterations, we end the process and conclude that 2 and -3 are roots of $x^2 + x - 6$. This is true if we evaluate the mathematical expression using algebra.

6 GA

Genetic Algorithm (GA) is an evolutionary algorithm developed by John Holland et. al. It is based on the mechanics of natural selection and natural genetics, that is, it imitates the processes involved in selection, recombination and evolution. It involves randomness due to the fact that it mimics natural processes, but users can control the degree of randomness that GA exhibits.

The goals of optimization seeks to improve performance towards some optimal point or points. However, there is a distinction between the process of improvement and the destination or optimum itself. In this case, GA is the process and is independent of the objective being approached. GA is not focused on solving a single problem. It is a flexible tool used under different circumstances. This robustness makes GA popular among optimization algorithms.

6.1 Short Background

GA was developed by John Holland with the help of his colleagues and students. Their goal was to (1) abstract and rigorously explain the adaptive process of natural systems and (2) design artificial system software that retains the important mechanisms of natural systems. This approach led to important discoveries in both natural and artificial systems.

6.2 Components of GA

The basic algorithm for GA is shown below in figure 8

6.2.1 Initialization of Population

As we can see, the first step is to generate a population sufficient enough to cover our search space and is limited by the resources at hand. Each member of this population is encoded as an array of values. The number of elements in the array will be determined by the problem and the one who creates the GA. The population size n determines how many chromosomes are in one generation. If there are too few chromosomes, GA will have a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down and many of the elements of the initial population are repeated. After several years of research, it was determined that after some limit (which depends mainly on encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster.[8] This is because the population size becomes too big for the solution space, hence, the population will only have many replicates.

6.2.2 Fitness Evaluation

Next is to evaluate the fitness function $f(x)$ for each chromosome x in the generation. A fitness function is the function that the algorithm is trying to optimize. The word "fitness"

Outline of the Basic Genetic Algorithm

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population
3. **[New population]** Create a new population by repeating following steps until the new population is complete
    1. **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
    2. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
    3. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
    4. **[Accepting]** Place new offspring in a new population
4. **[Replace]** Use new generated population for a further run of algorithm
5. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population
6. **[Loop]** Go to step **2**

Figure 8: Basic GA Algorithm[8]

is taken from the evolutionary theory. It tests and quantifies how 'fit' each potential solution is with respect to the problem.[9] It is important to note that the fitness function is a large factor in problem solving using GA. The fitness function must be able to define the numerical complexity and constraints that are present in the problem. Choosing the right fitness function will determine computability and complexity usage of the algorithm.

6.2.3 Selection

Selection allows for persistence and propagation of better genes in the next generation based on the current gene pool. The selection process is 'repeating' if it allows re-selection of already selected members. Selection is 'non-repeating' if it does not allow re-selection of members for cross-over. Non-repeating allows retention of other possibly 'good' genes (genes that might lead to better solutions later on) and a slower convergence rate. Repeating selection can lead to a population of individuals that have the same already good genes but differ in only some features. This allows for local exploration, searching for a good solution in a specific area in the search space. In consequence, since the same parents can be selected numerous times, it can lead to generating a population with a uniform genetic make-up.

Examples for selection process are roulette selection and elimination selection. Roulette selection is done by creating a roulette wheel where individuals that have better genes are provided with a lager portion of the whole wheel. The wheel is spun and the corresponding member mapped to the portion of the wheel where the pointer ends at is selected. The process is repeated until there is a good enough number for generating the next population. An example of the roulette wheel is seen on figure 9. Elimination selection is done by selecting a number of individuals and pitting them against each other based on their function values. Individuals that have better function values are selected and the process is repeated until there is a good enough number for generating the next population. An example of the elimination selection is seen on figure 10.

6.2.4 Recombination or Cross-over

Cross-over is the process of taking two selected individuals and swapping portions of their genetic make-up to create offspring that have genes from both parents (heredity). The number of points where crossing occurs is determined by the implementor. The cross-over chance is the probability that tells whether recombination occurs for a pair of chromosomes. Cross-over chance is determined by the implementor after some tests. Cross-over mechanism is important because it allows the creation of possibly new solutions from the previous gene pool. This allows exploration over a specific area in the search space. The individuals generated by this process are the members of the next generation. It is up to the implementer how much of the newly generated individuals are chosen. A possible implementation is where offspring that have the better function values may be retained. It is also possible to retain chromosomes form the previous generation. Suppose that we have chromosome A having the genetic make-up $< 1, 2, 3, 4 >$ and chromosome B having the genetic make-up $< 6, 7, 8, 9 >$. If we implement a single point cross-over after the second value we get the offspring $< 1, 2, 8, 9 >$ and $< 6, 7, 3, 4 >$. A visual representation is seen in figure 11.

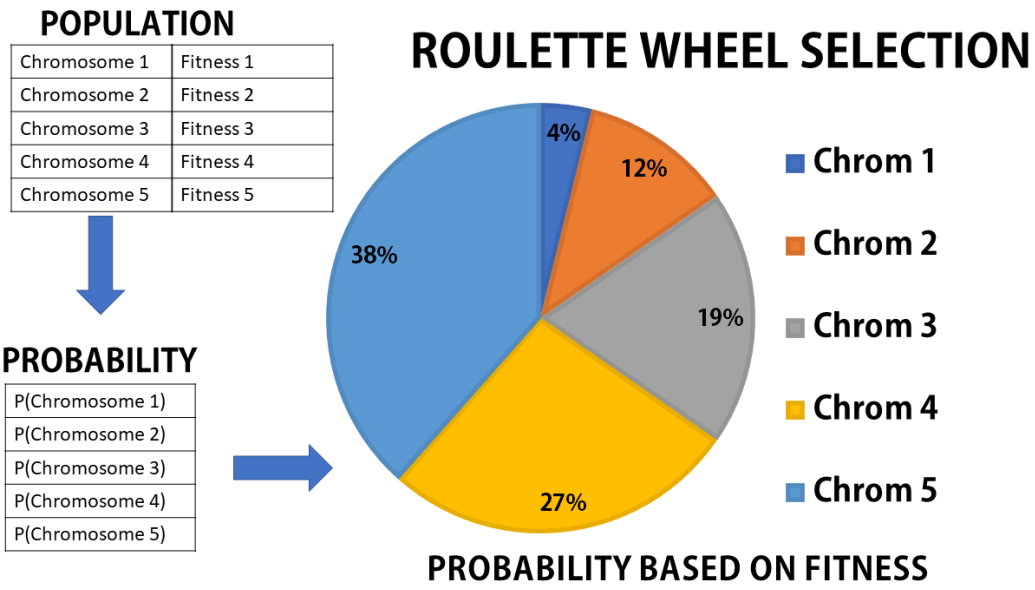

Figure 9: Roulette Wheel Selection

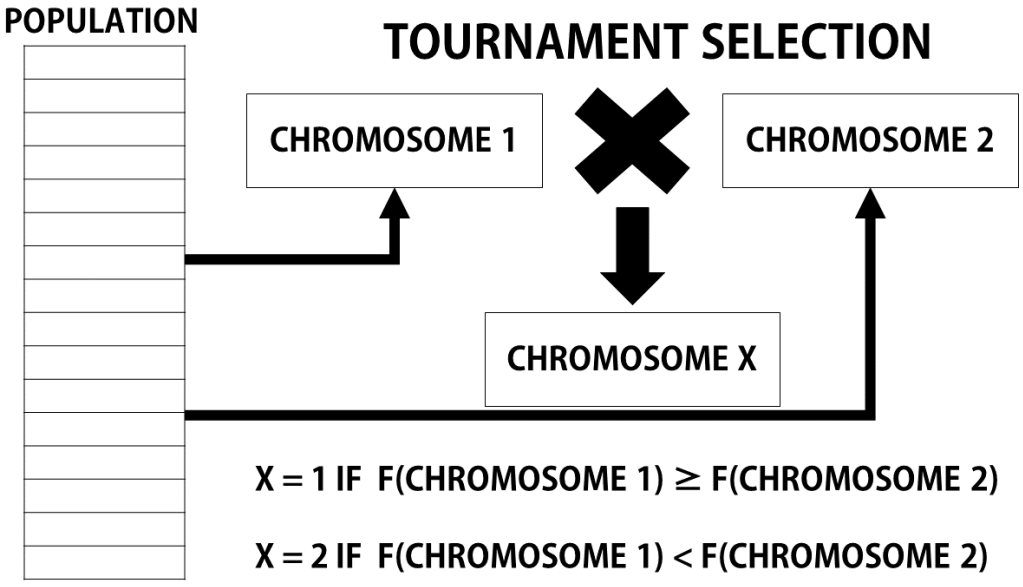

Figure 10: Simple Elimination Selection

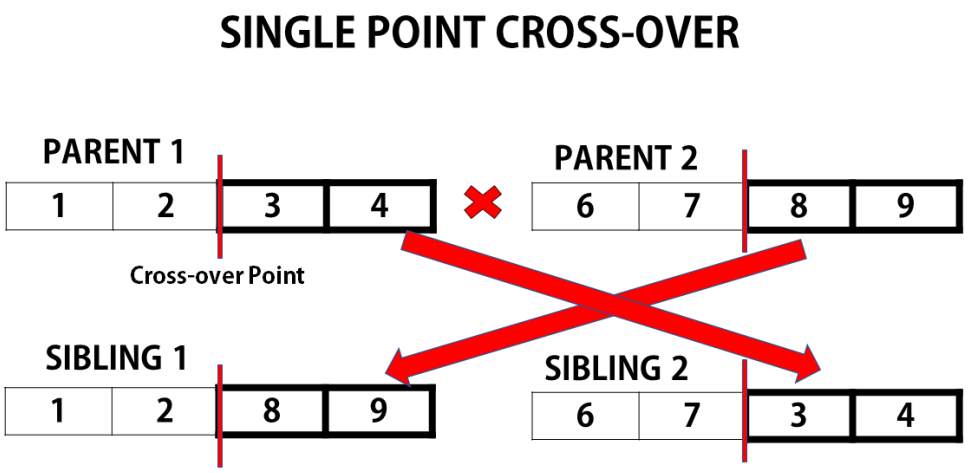

Figure 11: Singe Point Cross-Over

6.2.5 Mutation

Mutation mechanism is the process wherein some genes of members in the population are replaced by a completely new value. This allows for exploration on possibly 'unexplored' areas in the search space. It helps get the population unstuck from a local optima (optimum solution for a certain area in the search space but may not the most optimal of solutions in the entire search space). Mutation chance is determined by the implementor after some testing. In nature, however, mutation is a rare occasion hence the mutation chance must be low (usually 0.02). A visual representation is seen in figure 12. If the chromosomes are bound to have each gene $x_i \in [1, 9]$ where i is from $[1, 4]$. We can see that 3 is replaced by 9 and that both 3 and 9 are still in $[1, 9]$. Note that the number of genes (elements) to be mutated is not limited to one. You can change a few more genes but the number must be small. Mutation is stated to be some minor change in the genes this means that it is up to the implementor to determine the number genes to be manipulated such that it only brings a minor change. When we take binary numbers in consideration, flipping a few bits still creates a minor change if let's say that the chromosome is made up of 30 or 50 genes, then flipping 2-3 bits will not cause a major change provided that they are less significant bits.

6.2.6 Acceptance

Accepting is just evaluating whether or not the generated member can be added to the new population. This can be done through comparing with the parents' fitness values. If the offspring have better fitness values then accept, otherwise reject. This step us usually done after cross-over to check if the siblings generated will replace members in the population based on their fitness.

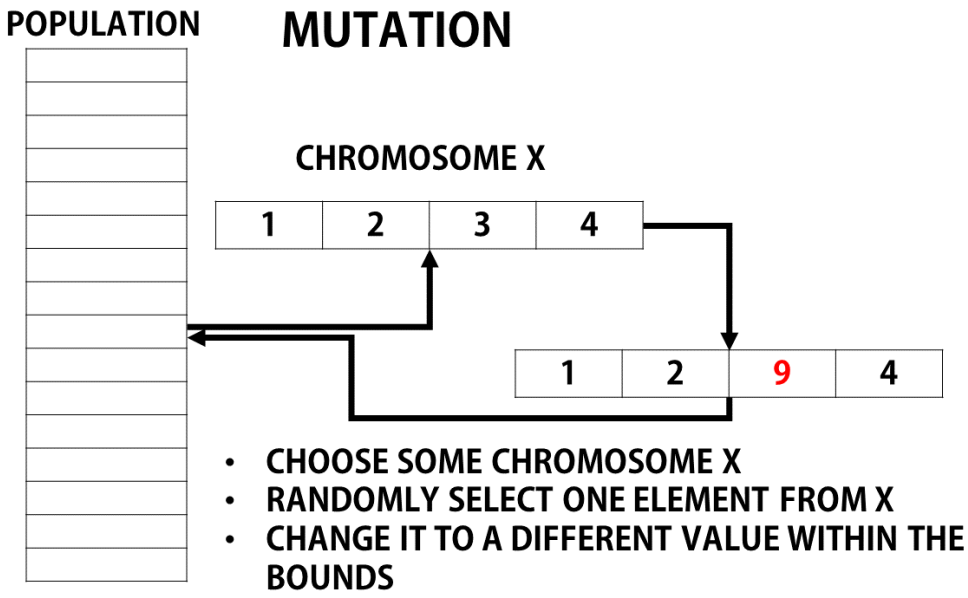

Figure 12: Mutation in GA

6.2.7 Replacement

Replace the old population with the new population. We can also choose to retain some of the old population, some of those that have 'good' genes can be kept in the new population. This is called being 'elitist' since it keeps only the fit members of society to move forward.

6.2.8 Termination Conditions

Testing is done through keeping track of the best fitness of each generation. If the fitness is the same for n amount of times and is below a certain acceptable threshold, then we terminate the process. This is considered as a success only if most of the members in the population have the same acceptable fitness, otherwise it is a failure. n is determined by the user. If the population becomes uniform, terminate the process and print out the value. If the fitness is acceptable under a threshold, then it is a success and we say that the population has converged to that point. If the population has become uniform but does not have an acceptable fitness, then it is a convergence but a failure. If a certain number of iterations has been reached and it has not yet converged and has been the same for n times, the process terminates and it is a failure.

6.3 Example

Consider again equation $F(x) = x^2 + x - 6$. We want to find the roots of $F(x)$, where $F(x) = 0$, using GA. Our fitness function is again $f(x) = |F(x)| = |x^2 + x - 6|$. The closer the value of $f(x)$ to 0, the better the x. We also define our search space as $x \in Z from[-7, 7]$ since we are using GA. We do this so that we can have the each integer converted to binary numbers with the most significant bit as sign.

We first generate a population of 10 members. We have the array Population[5][4] where the first column represents the sign of the number, 1 for negative and 0 for positive. The next 3 columns will contain the binary representation of each number. Note that we will have two representations of 0, namely, 1000 and 0000. We also have the array Fitness[5][1] where each row corresponds to each member's fitness value. We will also have the array TransitionPopulation[5][4] that will be used to hold the next generation population.

6.4 Initialization/Iteration 1

We randomly choose numbers from the set of integers from -7 to 7 for the location of each member. *Note that in this example, we will be using integers so that it will be easy to compute and visualize. Implementation is up to the user. Suppose we get the population:

Population[5][4]=

1	0	0	0
1	1	0	1
0	1	1	1
0	0	0	1
1	0	1	0

Next in our algorithm is to evaluate each chromosome's fitness under the fitness equation, $f(x)$. In order to do this, we simply translate our element chromosome into a decimal number. For chromosome 1, $x = 0$ as translated from binary to decimal, we calculate $f(x)$. $f(x) = |(x^2 + x - 6)| = |(0^2 + 0 - 6)| = | - 6| = 6$. We do this for the rest of the members and place the answers in respective rows in the Fitness array. We get:

Fintess[5][1]=

6
14
50
4
4

We record 4 as the best in this generation. Now we generate the new population by undergoing Selection, Cross-over, Mutation and Accepting. We select two members from the population and compare their fitness values. The better individual is chosen. We do this twice. Suppose we get the chromosomes ¡1,0,0,0¿ and ¡0,0,0,1¿ since ¡0,0,0,1¿ has a fitness value of 4 and ¡1,0,0,0¿ has a fitness value of 6, the member with the better fitness is selected it for Cross-over, that is ¡0,0,0,1¿. Now we choose another two and compare. Suppose we get ¡0,1,1,1¿ and ¡1,0,1,0¿ since ¡1,0,1,0¿ has a better fitness value, we select it. Now that we have two parents for Cross-over, we do a single-cross after the 2nd element. We produce the offspring ¡0,0,1,0¿ and ¡1,0,0,1¿ which have the fitness values 0 and 6 respectively. Comparing the two, we only select ¡0,0,1,0¿ for our next generation. We do this for 4 more times and suppose we end up with:

TransitionPopulation[5][4]=

0	0	1	0
1	0	0	1
0	0	1	1
0	0	1	0
1	0	1	1

Now we choose one random gene and mutate it (flip the value). Suppose we choose the gene TransitionPopulation[5][1] it will become 0.
Now out new TransitionPopulation is:

TransitionPopulation[5][4]=

0	0	1	0
1	0	0	1
0	0	1	1
0	0	1	0
0	0	1	1

We have already talked about the stopping criterion previously. For this case, our goal is that if $f(x)$ is equal to 0 for 3 iterations since we are only considering integers, we are sure that 0 will be found. This will be judged as a success.
We choose that our maximum iteration as 10 and if that is reached, the simulation has failed. Since not all members' fitness values are not near the goal, and we have just started, we loop the process.

6.5 Iteration 2

We now replace the old generation with the new generation by replacing all the elements in Population with the elements in TransitionPopulation.

Population[5][4]=

0	0	1	0
1	0	0	1
0	0	1	1
0	0	1	0
0	0	1	1

We evaluate each chromosome's fitness under the fitness equation, $f(x)$. We get:

Fintess[5][1]=

0
6
6
0
6

We record 0 as the best in this generation. Now we generate the new population by undergoing Selection, Cross-over and Mutation.

TransitionPopulation[5][4]=

0	0	1	0
0	0	0	1
0	0	1	0
1	0	1	1
0	0	1	0

Since not all members' fitness values are not near the goal, we loop the process.

6.6 Iteration 3

We now replace the old generation with the new generation by replacing all the elements in Population with the elements in TransitionPopulation.

Population[5][4]=

0	0	1	0
0	0	0	1
0	0	1	0
1	0	1	1
0	0	1	0

We evaluate each chromosome's fitness under the fitness equation, $f(x)$. We get:

Fintess[5][1]=

0
4
0
0
0

We record 0 again as the best in this generation. So now it is twice in a row. One more and it will be a success. Now we generate the new population by undergoing Selection, Cross-over and Mutation.

TransitionPopulation[5][4]=

1	0	1	1
0	0	1	0
1	0	1	0
0	0	1	0
0	0	1	0

Since not all members' fitness values are not near the goal, we loop the process.

6.7 Iteration 4

We now replace the old generation with the new generation by replacing all the elements in Population with the elements in TransitionPopulation.

$$\text{Population}[5][4]= \begin{array}{|c|c|c|c|} \hline 1 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline \end{array}$$

We evaluate each chromosome's fitness under the fitness equation, $f(x)$. We get:

$$\text{Fintess}[5][1]= \begin{array}{|c|} \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline \end{array}$$

We record 0 again as the best in this generation. Since this is the third, we will terminate this process. Now we generate the new population by undergoing Selection, Cross-over and Mutation.

$$\text{TransitionPopulation}[5][4]= \begin{array}{|c|c|c|c|} \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Since all members' fitness values are at the goal, we stop process. We get the values 2 and -3 as the solutions to $f(x)$ which is the same as in PSO.

7 CODEQ

CODEQ is an evolutionary algorithm proposed by Omran[10]. It is a population-based meta-heuristic method for global optimization designed for continuous spaces. It is based on four concepts, namely, Differential Evolution, Opposition-based Learning, Chaotic Search and Quantum Mechanics.

7.1 Differential Evolution

Differential Evolution is an evolutionary population-based algorithm proposed by Storn and Price [11]. It is similar to any other evolutionary algorithm except that distance and direction information is used to guide the population, in contrast to GA which guides the population using fitness standards. DE uses the differences between randomly chosen vectors (individuals) as the source of random variations for a third vector (new individual), referred to as the target vector. Trial solutions are generated by adding weighted difference vectors to the target vector. This process is referred to as the mutation operator where the target vector is mutated. A recombination, or crossover step is then applied to produce an offspring which is only accepted if it improves on the fitness of the parent individual. For reference, this crossover is similar to the that of GA.

The basic DE algorithm is described in more detail below with reference to the three evolution operators: mutation, crossover and selection. *Mutation*: For each parent $x_i(t)$, of generation t, a trial vector $v_i(t)$ is created by mutating a target vector. The target vector $x_{i_3}(t)$ is randomly selected, with $i \neq i_3$. Then two individuals $x_{i_1}(t)$ and $x_{i_2}(t)$ are randomly selected with $i \neq i_1 \neq i_2 \neq i_3$, and the difference vector, $x_{i_1}(t)$-$x_{i_2}(t)$, is calculated. The trial vector is then calculated as

$$v_i(t) = x_{i_3}(t) + F(x_{i_1}(t) - x_{i_2}(t))$$

where the last term represents the mutation step size. F is a scale factor used to control the amplification of the differential variation. Note that $Fin(0, \inf)$.
Crossover: DE follows a discrete recombination approach where elements from the parent vector $x_i(t)$, are combined with elements from the trial vector, $v_i(t)$ to produce the offspring $\mu_i(t)$. Using the binomial crossover,

$$\mu_i ij(t) = \begin{cases} v_{ij}(t), & \text{if } \bigcup(0,1) < P_r \text{ or } j = r \\ x_{ij}, & \text{otherwise} \end{cases}$$

where $j = 1, ..., N_d$ refers to a specific dimension, N_d is the number of genes (parameters) of a single chromosome, and $r \bigcup(1, ..., N_d)$. In the equation above, P_r is the probability of reproduction (with $P_r \in [0, 1]$). In example, if chromosome A is [0,1,2,3,4] and chromosome B is [5,6,7,8,9], there are a lot of possible combinations for the output such as [0,6,2,3,9] and [5,1,2,8,4]. Unlike the single crossover seen in GA where there are only two possible outcomes. This is because each gene is individually tested on whether or not they would cross
Thus, each offspring is a stochastic linear combination of three randomly chosen individuals when $\bigcup(0, 1) < P_r$; otherwise the offspring inherits directly from the parent. Even when $P_r = 0$, at least one of the parameters of the offspring will differ from the parent (forced by the condition $j = r$).
Selection: DE evolution implements a very simple selection procedure. The generated offspring, $\mu_i(t)$, replaces the parent, $x_i(t)$, only if the fitness of the offspring is better than that of the parent.
Empirical studies have shown that DE performance is sensitive to its control parameters [12, 13]. Another program with DE is that it is not rotationally invariant (i.e. The performance of DE depends on the orientation of the coordinate system in which the objective function is evaluated). The main reason for this problem is the crossover. Only when $P_r = 1$ (only mutation no crossover), the DE becomes rotationally invariant [14]. Because DE is not rotationally invariant, empirical results show that it struggles with non-separable function [**?**].

7.1.1 Visualizing DE

So we first discuss why there are distances and direction when we are talking about points in space. It is simple, we are talking about vectors and not mere points in space. Each individual in the population is a vector that has the same origin (0,0). Subtracting two vectors v_1 and v_2 will result in a vector v_3 that gives the distance generated between the two

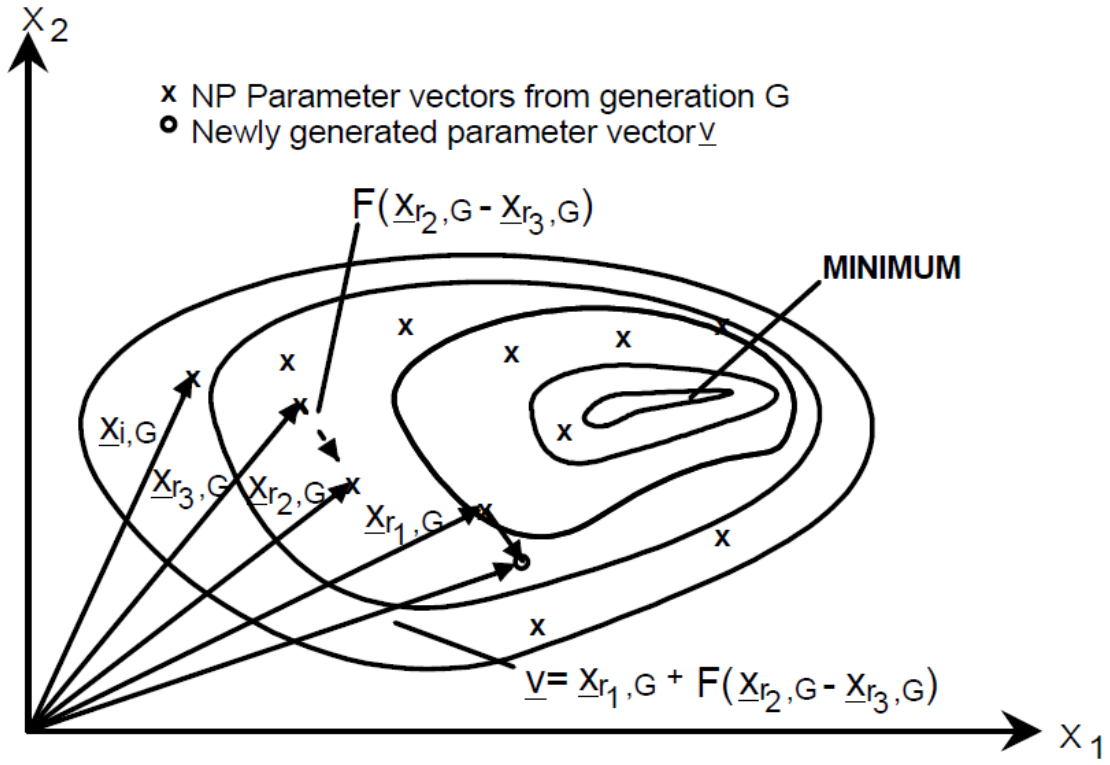

Fig.1: Two dimensional example of an objective function showing its contour lines and the process for generating v in scheme DE1.

Figure 13: Visual of the process used in DE[11]

vector heads. The direction of the resulting vector v_3 is determined by which vectors are the subtrahend and minuend as seen on the figure 5 in section 1.5. A visualization of how the vectors are used in DE is seen on figure 13, taken from by Storn and Price[11]. They also gave a visual on how the cross-over operator works on figure 14.

7.1.2 DE Algorithm

The algorithm for DE is presented as follows:

1. Generate an initial population X with N members

2. For each member i ($i = 0, 1, ..., N$) in the population, evaluate objective function values for $F(x)$

3. Record the best overall function value.

   If record does not exist yet, set best $F(x)$ of initial population to be the current best.

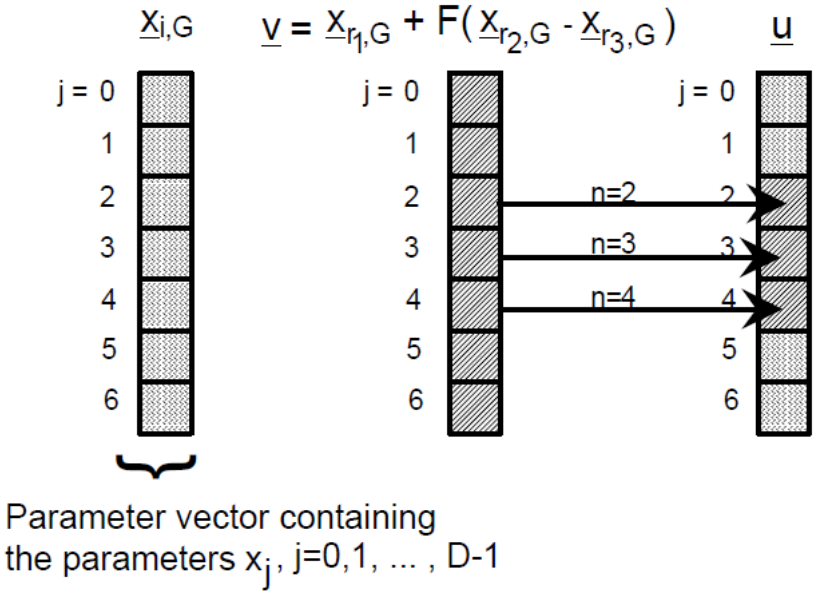

Figure 14: Illustration of the Crossover Mechanism in DE[11]

Else compare the recorded best overall with the best from the current population. Set the record to whichever value is better (smaller for minimization).

4. To generate the next population, for each member i designated as x_{i_3},

   (a) Generate 2 random unique integers from $[0, N-1]$ (not equal to the index of x_{i_3}) to obtain x_{i_1}, x_{i_2}

   (b) Perform Mutation by using the equation:

$$v_i(t) = x_{i_3}(t) + F(x_{i_1}(t) - x_{i_2}(t))$$

   (c) Perform crossover on v_i and x_{i_3} based on a probability P_r to obtain μ_i using:

$$\mu_i ij(t) = \begin{cases} v_{ij}(t), & \text{if } \bigcup(0,1) < P_r \text{ or } j = r \\ x_{ij}, & \text{otherwise} \end{cases}$$

   (d) Perform selection/decision by checking if $F(\mu_i) < F(x_{i_3})$.

   If $F(\mu_i) < F(x_{i_3})$, replace x_{i_3} with μ_i

   Else retain x_{i_3} for the next generation

5. After the next generation is obtained, repeat from 2 until maximum generations is reached.

7.2 Opposition-based Learning

Opposition-based learning (OBL) was first proposed by Tizhoosh[15] and was succesfully applied to several problems by Tizhoosh et. al.[16] The basic concept of OBL is consideration of an estimate and its corresponding opposite estimate simultaneously to approximate the current candidate solution.
Opposite number are defined as follows: Let $x \in [a, b]$, then the opposite number x' is defined as

$$x' = a + b - x$$

The above defintion can be extended to higher dimensions as follows: Let $P(x_1, x_2, ..., x_n)$ be an n-dimensional vector, where $x_1 \in [a_1, b_1]$ and $i = 1, 2, ..., n$. The opposite vector of P is defined by $P'(x'_1, x'_2, ..., x'_n)$ where

$$x'i = a_i + b_i - x_i.$$

7.2.1 Visualization and Faults of OBL

OBL uses proximity to reduce the search space. If one number k and its opposite k' are both in the search space, the one closer to a solution will be selected, and the search space is reduced by half. This method is similar to the Bisection Method. The proximity is dictated by an objective function. Hence, if $f(k) > f(k')$ then k is selected and half the search space where k' is located becomes 'cut' or rendered un-explorable.
If we are dealing with a population of N members, all members located in the search space that was 'cut' will be reflected onto the uncut search space by using their opposite numbers. The 'cut' space will be determined by the best individual, that member which obtained the best function value $f(x)$.
A single point on a linear search space is given as an example by Tizhoosh[15] in figure 15. As we can see from the figure, the point x is randomly generated and is visually placed on the line during $k = 1$. Its opposite xo is also projected on the same line. Given that x is closer to the solution, we 'cut' the half of the domain by which xo is located. In the next step, the space where x is located is then cut because this time, xo was projected closer to the solution. Note that the projection of x changes because the limits $a1$ and $b1$ changes as we reduce the search space. If we continue this trend, we will end up having a small margin in which x and xo are so close that the space in between does not matter, hence obtaining an approximate to the solution or the solution itself when $x = xo$.
The dangers of using such method is revealed when we have multiple optima, as well as having local optima. The search space may be reduced such that the global optima is 'cut' from the explorable search space. Another problem arises when the solution is equidistant from both x and xo. In this situation, we may arbitrarily choose which side to 'cut', however if the solution space is not continuous then a problem may arise, hence, it is mandatory that the search space used is continuous.

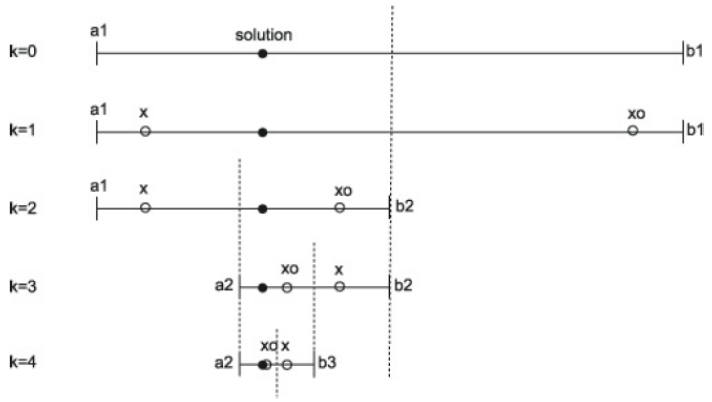

Figure 1. Solving a one-dimensional equation via recursive halving of the search interval with respect to optimality of the estimate x and opposite-estimate xo.

Figure 15: Illustration of OBL process for a linear search space[15]

7.3 Chaos Search

Chaos is a characteristic of non-liner system that includes infinite unstable periodic motions and depends on initial conditions. Due to its uncertainty, ergodicity and stochastic properties, chaotic sequences have been used to replace random numbers and to enhance the performance of heuristic optimization algorithms such as GA, PSO and others. There are several chaotic maps with different ergodic properties and spread-spectrum characteristics. The piecewise linear chaotic map (PWLCM) is a simple and efficient chaotic map with good dynamic behavior. The simplest PWLCM is defined by Xiang, Liao and Wong [17],

$$x(t+1) = \begin{cases} x(t)/p, & x(t) \in (0,p) \\ \frac{1-x(t)}{(1-p)}, & x(t) \in [p,1) \end{cases}$$

The PWLCM behaves chaotically in (0,1) when $p \in (0.05) \bigcup (0.5,1)$. The chaotic variable, x, can be randomly initialized (i.e. $x(0) \bigcup (0,1)$) as suggested by Xiang et.al [17] who implemented the PWLCM in PSO to perform chaotic search. They implemented the CPSO (Chaotic PSO) by adding the term $r(2cx-1)$ to the global best $\hat{y}$. cx is the chaotic variable given by PWLCM and r is a random number taken from the uniform distribution of $(0,1)$. If the resulting vector's objective function value is better, then the global best is replaced, if not, then retain the global best. The velocity function they used for this method is quite different as they have taken inspiration from Clerc and Kennedy [18]:

$$v_{ij} = \chi(v_{ij} + c_1 r_1(y_{ij} - x_{ij}) + c_2 r_2(y_j - x_{ij}))$$

Although it is not very different from the equation of Kennedy et.al.[1], there is no inertial weight present but the variable χ (a.k.a Constricting Factor) is new. χ is added so that the velocity of a particle is throttled such that it does not fly too fast (not having too high of

a magnitude for a single time/generation step). χ replaces the need of having to manually set a bound on the magnitude of the velocity. To recall, the velocity of a particle in PSO is usually set to have a bounded magnitude so that it does not travel too fast through the search space, thereby adding realism and further enhance the ability of each particle to explore the search space thoroughly.

7.4 Quantum Mechanics

In Newtonian mechanics a particle has a position and a velocity that determines its trajectory. However, in quantum mechanics, the particle's position and velocity cannot be determined simultaneously according to the uncertainty principle. Hence, the term trajectory is meaningless[19]. Sun et.al.[19] proposed a quantum model of PSO, called QPSO, where particles move according to the following equation,

$$x(t+1) = g \pm \frac{L}{2} ln(\frac{1}{\mu})$$

where g is a local attractor, L is a parameter that must go to zero as $t- >$ inf to guarantee convergence and $\mu \bigcup(0,1)$. L is a very important parameter of QPSO and different methods have been proposed to determine it.[19, 20]
Uncertainty principle states that the position and the velocity of an object cannot both be measured exactly, at the same time, even in theory. The very concepts of exact position and exact velocity together, in fact, have no meaning in nature. The uncertainty principle implies that there is no exact trajectory since there is no absolute measurement to the position and/or velocity of an object. This is because there will always be an unknown amount in the measurement given by the precision of the instruments used.

8 Constrained Optimization Problems

A constrained optimization problems is a problem that is bounded by some limiting factors. Typically, real-worlds problems are always subjected to constraints. For example, if you and your partner were to each create 3d models of a cube with a single piece of 10x15 inches cardboard. Given that you have limited resources, what is the best way to cut the cardboard in order to have 2 models with the least amount of unused cardboard? That was just a simply problem now suppose you have a lot of constraints, the problem will be more difficult as more constraints are added. Not only the quantity but also the type of the constraints affect the problem as well.

Most real-world optimization problems have constraints of different types which modify the shape of search space. During the past years, optimization algorithms have been employed to solve such problems. Constraints can be in the form of both equalities and inequalities, they can be discrete and continuous, linear or non-linear, they can also be related to other constraints. Due to these properties, constrained optimization problems are more difficult to solve compared to unconstrained ones. Garg[21] defines Constrained optimization

problems in figure 16.

With that explained, we go on to discuss what feasible and infeasible solutions are. Feasible

A general non-linear constrained optimization problem is defined as follows.

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{subject to} \quad & h_k(x) = 0 \quad ; \quad k = 1, 2, \ldots, p \\ & g_j(x) \leqslant 0 \quad ; \quad j = 1, 2, \ldots, q \\ & l_i \leqslant x_i \leqslant u_i \quad ; \quad i = 1, 2, \ldots, n \end{aligned} \tag{1}$$

where $x = [x_1, x_2, \ldots, x_n]^T$ denotes the n-dimensional vector of decision variables; f is the objective function; l_i and u_i are the minimum and maximum permissible values for the ith variable respectively; p is the number of equality constraints and q is the number of inequality constraints. Let $S = \{x \in \mathbb{R} \mid g_j(x) \leq 0; j = 1, 2, \ldots, p + q(= M)\}$ be the set of the feasible solutions, so the problem (1) can be formulated as:

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{subject to} \quad & g_j(x) \leqslant 0 \quad ; \quad j = 1, 2, \ldots, M \\ & l_i \leqslant x_i \leqslant u_i \quad ; \quad i = 1, 2, \ldots, n \end{aligned} \tag{2}$$

Figure 16: Constrained Algorithms Definition by Garg

solutions are solutions that do no violate any constraint while infeasible solutions do. There are many approaches in considering feasible and infeasible solutions when implementing optimization algorithms. Some of these methods include rejection of infeasible individuals, maintaining a feasible population, repairing of infeasible individuals, separation of individuals and constraints, replacement of individuals by their repaired versions and use of decoders.[22]

8.1 Penalty Function Approach

In order to solve constrained optimization problems, one may use penalty functions. In using penalty functions, the number of constraint violations are used to punish infeasible solutions so that feasible solutions are much more favored. Unfortunately, penalty functions require parameter tuning for different problems because these parameters are problem-specific.

He and Wang[23] utilized penalty functions for their Co-evolutionary PSO implementation. They used two groups of swarm(s). The first group is used to explore the search space while the other group is used to tweak the penalty function parameters. Each swarm in the exploration group is paired with an individual in the parameter group. The individuals in the parameter group determine the penalty functions to be used by the corresponding swarms in the exploration groups. Hence, the solutions obtained in the exploration group depend upon the parameter group while the parameter group depend on the exploration group for evaluation and tweaking. The process aimed to explore and exploit different search spaces in finding the solution.

On the other hand, Deb[24] proposes a parameter free function in creating a better population to solving constrained optimization problems using GA. These penalty functions do not require values to be set by the user instead, it utilizes the values of the constraint violation themselves. Parameter free penalty function is driven by the new fitness value system in figure 17.

search space S using a modified objective function F such as

$$F(x) = \begin{cases} f(x) & \text{if } x \in S \\ f_w + \sum_{j=1}^{M} g_j & \text{if } x \notin S \end{cases}$$

where x is the set of solutions obtained by approaches and f_w is the worst feasible solution in the population.

Figure 17: Fitness Function using Penalty Functions

If the individual from the population satisfies all conditions, it's fitness value is unchanged but if it does not satisfy the conditions, it's fitness value is changed to that of the worst value added with the values of the violated inequality constraints g. This allows the selection operator to give a better chance to feasible solutions by setting the fitness values of infeasible solutions far away from the objective. In PSO and PSO-GA, infeasible solutions means that the population ignores them and only flock towards feasible solutions. In GA, infeasible solutions are ignored or have the least chance of being selected for the selection process.

8.2 PSO Fly-back Approach

One method for keeping feasible solutions is to make the particles return to their previous positions.[25] When the individual is to venture upon the infeasible solutions, it "moves back" to its previous position instead of flying to the infeasible solution space. This is done be simple retaining the position it currently is at. One the other hand, while the position remains unchanged, the change in velocity is retained hence, in the next iteration, the particle's velocity becomes shorter and more attuned to facing towards the global best position. This method retains a feasible population but the initial population must be feasible.

9 PSO-GA

PSO-GA is a hybrid of PSO and GA. Harish Garg has proposed a PSO-GA[21] which supplements the particular disadvantages of both PSO and GA with the advantages of each. The algorithm attempts to balance the exploration and exploitation ability of both algorithms. Exploration happens in PSO when particle fly through the search space. It is less applicable to GA since the algorithm only utilizes what is current known in the population. It only occurs for GA through Cross-over and Mutation. Exploitation happens in PSO when a particle flies to or near an area containing a possible solution, every other particle in the population will tend to flock towards that area in order to find the solution. PSO's problem is that local optima may trap the whole population. Exploitation happens in GA during the Selection operator, wherein the members with the fittest values have a higher chance of being chosen for Cross-over and Mutation. Hence, more chances of exploring that particular gene pool.

In GA, if an individual is not selected, the information contained by that individual is

lost but in PSO, the memory of the previous best position is always available to each individual. Without a selection operator, PSO may waste resources on poorly located individuals. PSO-GA by Garg[21] combines the ability of social thinking in PSO with the local search capability of GA.
PSO's velocity vector guides the population to a certain solution point while GA's selection and cross-over replaces infeasible solutions with feasible ones by creating an individual from the set of feasible solutions.

9.1 Parts of PSO-GA

The basic algorithm for PSO-GA is shown in figure 18. The notations of the algorithm is given in figure 19. The equations needed for some variables in the algorithm is given in figure 20.

As you can see, the algorithm follows the both PSO and GA algorithms in succession. PSO is first done to the population to obtain points across the search space. GA is then applied to some of the best individuals. This is done to replace the worst individuals in the population with those closer to the better ones.

After forming the new population with PSO, some of the individuals in the population will get replaced. Some not all because if we have a huge population, it would take a long time to complete. This number is given by GA_{Num} in figure 20. After selecting the best individuals from the population, the algorithm aims to create a new population by replacing points in the current population with better points via the genetic principles, selection, cross-over and mutation. After all selected individuals have been processed, we change the GA variables, GA_{PS} and $GA_{MaxIter}$ which are for the population size in GA and the maximum iterations done for GA respectively by the equations in figure 20.

Judging from the equations in figure 20, GA_{Num} will initially be GA_{NumMax} and slowly become GA_{NumMin} as the number of iterations increases. This is because the fraction $PSO_i/PSO_{MaxIter}$ is raised to γ which is a positive whole number as given by Garg[21], hence, the whole term $(PSO_i/PSO_{MaxIter})^{\gamma}$ will initially be very small and eventually will be equal to 1 when $PSO_i = PSO_{MaxIter}$.

This is also the case for both GA_{PS} and $GA_{MaxIter}$. GA_{PS} will initially start equal to GA_{MinPS} then slowly become GA_{MaxPS}. $GA_{MaxIter}$ will initially start equal to $GA_{MinIter}$ then slowly become $GA_{MaxIter}$. Since the factors will be in fractions, there is a need to get the floor values of GA_{Num}, GA_{PS} and $GA_{MaxIter}$. This is because GA_{Num}, GA_{PS} and $GA_{MaxIter}$ must be positive integers because they dictate array sizes. However, in the case of Inertial Weight w, which changes according to the equation $w = w_{max} - (w_{max} - w_{min})(PSO_{iter}/PSO_{MaxIter})$, it is most of the time a fraction. Garg[21] started $w = 0.9$ initially then becoming $w = 0.4$ as the number of iterations increases.

9.2 Example

Consider again the equation $F(x) = x^2 + x - 6$. We already know that to get $F(x) = 0$, $x = 2$ or $x = -3$. We will be looking at 1 iteration in depth just to have a clear understanding of what happens for the entire algorithm. Our fitness function is still $f(x) = |F(x)| = |x^2 + x - 6|$. We also define our search space as $x \in [0, 15]$ but we will only consider integers.

This is so that the numbers can be represented in binary so that we can implement it for GA. Take note that it will only be possible to get the root $x = 2$ for this problem.

We first generate a population of 5 members. We have the array PSOPopulation[5][4] where each row is a member that is in 4 bit binary format. We have the array Velocity[5][5] where the first column represents the sign (0 for positive, 1 for negative) and the remaining columns represent the 4 bit binary representation of the velocity. Take note that in this format we have 2 representations for 0, that is $< 0, 0, 0, 0, 0, 0 >$ and $< 1, 0, 0, 0, 0, 0 >$ which is 0 and -0. Next we have our PSOFitness[5][1] which holds our fitness values of our population for the PSO part of the process. We also have Pbest[5][5] where the first column is the personal best fitness of each member and the rest hold the 4 bit binary representation of the best position. We also have Gbest[1][5] where the first column is the personal best fitness overall and the rest hold the 4 bit binary representation of the best position overall. We also have an array PSOTransPopulation[5][4] that will hold the values of the next locations.

9.2.1 ITERATION 1

We randomly choose numbers from the set of integers from 0 to 31 for the location of each member. We also randomly choose numbers from the set of integers from -15 to 15 for the initial velocity of each member. Suppose we get the population with members 4, 10, 15, 30 and 21:

PSOPopulation[5][4]=

0	1	0	0
1	0	1	0
1	1	1	1
1	1	0	0
0	1	0	1

and the velocities, +5, -2, -3, +1, +6:

Velocity[5][5]=

0	0	1	0	1
1	0	0	1	0
1	0	0	1	1
0	0	0	0	1
0	0	1	1	0

We first evaluate the fitness so that we can generate our initial Pbest and Gbest values. For the first member, $< 0, 1, 0, 0 >$ which is 4 in decimal. Evaluating it for the equation $F(4) = |4^2 + 4 - 6| = 14$. We do this for the rest. We get

PSOFitness[5][1]=

14
104
219
138
24

Now we record the Pbest and Gbest values. Since this is the initial step, we just copy from the PSOFitness and PSOPopulation for each member for Pbest. As for the Gbest since $x_0 = 4$ gave us the smallest fitness value, we copy its data. Hence, we have:

Pbest[5][5]=

14	0	1	0	0
104	1	0	1	0
219	1	1	1	1
138	1	1	0	0
24	0	1	0	1

Gbest[1][5]=

14	0	1	0	0

Next we calculate the new velocities of each member by the given equation in our PSO algorithm. We first choose $c1 = c2 = 2$. We need to implement randomly generated numbers from 0 to 1 but since we are simulating, we choose it to be 0.5 for all members. We also implement inertial weight $w = 1$. For member 1, $x_0 = 4$ and $v_0 = 5$. We get $v_0 = (w)(v_0)+(2)(0.5)(Pbest[0][4]-Population[0][4])+(2)(0.5)(Gbest[2:4]-Population[0][4]) = (1)(5) + (2)(0.5)(4-4) + (2)(0.5)(4-4) = 5 + (0) + (0) = 5$. For member 2, $x_1 = 10$ and $v_1 = -2$. We get $v_1 = -2 + (2)(0.5)(10-10) + (2)(0.5)(4-10) = -2 + (0) + (-6) = -8$. We do this for the rest and get:

Velocity[5][5]=

0	0	1	0	1
1	1	0	0	0
1	1	1	1	0
1	0	1	1	1
0	0	1	0	1

We now get the new population by adding the velocities to our position population. We save it first in TransPopulation so that we can see the difference later. For the first member, $x_0 = 4$, since the velocity is still $v_0 = 5$, we just add them to get $x_0 = 9$. For the second member, $x_1 = 10$, since the velocity is $v_1 = -8$, we add them and get $x_1 = 2$. We do this for the rest and get:

PSOTransPopulation[5][4]=

1	0	0	1
0	0	1	0
0	0	0	1
0	1	0	1
1	0	1	0

9.2.2 ITERATION 1 for GA part of PSO-GA

We now do the GA part of the PSO-GA. Let us first suppose that GA_{PS} or the population size of the GA is 5, that is, it is the same size with our PSO population. Suppose also that GA_{Num} is 2, that is only 2 members of the PSOTransPopulation will be selected and replaced. We also suppose that GA_{MaxItr} is 3 since we are only trying to simulate the process. We must generate a random population again from $[0, 31]$ and store them in the array

GAChromosomes[5][4]. From the algorithm, the first element in our GA population should be selected among the best individuals from PSOTransPopulation. In order to do this, we must get the fitness values of the PSOTransPopulation and store them in PSOTransFitness[5][1]. We have:

$$\text{PSOTransFitness[5][1]}= \begin{array}{|c|}\hline 84 \\ \hline 0 \\ \hline 4 \\ \hline 24 \\ \hline 104 \\ \hline \end{array}$$

We know that the position with the best value in ou PSOTransFitness is 2. So we have:

$$\text{GAChromosome[5][4]}= \begin{array}{|c|c|c|c|}\hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 \\ \hline \end{array}$$

We then calculate the fitness of our GA population and store them in GAFitness. We use the same fitness function $f(x) = |F(x)| = |x^2 + x - 6|$ for our evaluation. We get

$$\text{GAFitness[5][1]}= \begin{array}{|c|}\hline 0 \\ \hline 4 \\ \hline 50 \\ \hline 104 \\ \hline 66 \\ \hline \end{array}$$

We keep the elite, that is we keep only the 3 (arbitrary) best members of our population and the rest will be generated by cross-over and mutation. So our GAChromosome[5][4] will be:

$$\text{GAChromosome[5][4]}= \begin{array}{|c|c|c|c|}\hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 \\ \hline - & - & - & - \\ \hline - & - & - & - \\ \hline \end{array}$$

Now we form our two other elements through cross-over and mutation. For simplicity, suppose we select $< 0, 0, 1, 0 >$ and $< 1, 0, 0, 0 >$ as the parents. We cross them after the second element and obtain $< 0, 0, 0, 0 >$ and $< 1, 0, 1, 0 >$. For our mutation, we just select the first sibling, and mutate one of its elements. Therefore we get, $< 0, 0, 0, 1 >$. Hence we now use these two for our population. We therefore have:

$$\text{GAChromosome[5][4]}= \begin{array}{|c|c|c|c|}\hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 1 & 0 \\ \hline \end{array}$$

Suppose that after 2 more iterations, the GA population has become

$$\text{GAChromosome}[5][4]= \begin{array}{|c|c|c|c|}\hline 0&0&1&0\\\hline 0&0&0&1\\\hline 0&0&1&0\\\hline 0&0&0&1\\\hline 0&0&1&1\\\hline\end{array}$$

We know that $< 0, 0, 1, 0 >$ will give us the best value out of all of these. Therefore we replace the worst member for our PSOTransPopulation with $< 0, 0, 1, 0 >$.
Since, $< 1, 0, 1, 0 >$ in our PSOTransPopulation gave the worst fitness value, we replace it with $< 0, 0, 1, 0 >$. Hence our TransPopulation will be

$$\text{PSOTransPopulation}[5][4]= \begin{array}{|c|c|c|c|}\hline 1&0&0&1\\\hline 0&0&1&0\\\hline 0&0&0&1\\\hline 0&1&0&1\\\hline 0&0&1&0\\\hline\end{array}$$

We again do the same GA process with the second best in our PSOTransPopulation which is $< 0, 0, 0, 1 >$. We will replace $< 1, 0, 0, 1 >$ because it gave the second worst value. Suppose we got $< 0, 0, 1, 0 >$ from our second GA pass. Our PSOTransPopulation becomes:

$$\text{PSOTransPopulation}[5][4]= \begin{array}{|c|c|c|c|}\hline 0&0&1&0\\\hline 0&0&1&0\\\hline 0&0&0&1\\\hline 0&1&0&1\\\hline 0&0&1&0\\\hline\end{array}$$

We replace our PSOPopulation with our PSOTransPopulation and get

$$\text{PSOPopulation}[5][4]= \begin{array}{|c|c|c|c|}\hline 0&0&1&0\\\hline 0&0&1&0\\\hline 0&0&0&1\\\hline 0&1&0&1\\\hline 0&0&1&0\\\hline\end{array}$$

Clearly, it is seen that with just one iteration, the population has become better. If we continue to repeat this process, we will end up with a converged population. That is, the population becomes

$$\text{PSOPopulation}[5][4]= \begin{array}{|c|c|c|c|}\hline 0&0&1&0\\\hline 0&0&1&0\\\hline 0&0&1&0\\\hline 0&0&1&0\\\hline 0&0&1&0\\\hline\end{array}$$

We have solved the objective function and obtained the root $x = 2$. The other root is not obtained because we only considered positive integers from $[0, 31]$.

10 Methodology

10.1 Implementation

We first create a simple PSO code and a simple GA code then we combine both to create the PSO-GA. The presented algorithms are implemented in Matlab (MathWorks) and the program has been run on an Intel®Core™i5-6200U 2.3GHz processor with 16 GB of Random Access Memory (RAM).

10.2 Testing

We will first test our 3 algorithms for the function $f(x) = X^2$. This is a fairly easy problem to solve. We already know that by using algebra, the minimum value obtainable is 0.

We have our solution space to be from $[0, 31]$, integers only so that we can easily encode them into binary representations. We will test each algorithm 10 times each. For PSO, we will have $c1 = c2 = 1.5$ and w from 0.9 to 0.4 depending on the current iteration, as implemented in PSO-GA algorithm. For GA, we will have 0.85 for our cross-over probability and 0.02 for our mutation probability. For PSO-GA, we will have the same values for our c1, c2, w, cross-over probability, and mutation probability. For all algorithms, we will have a maximum of 1000 generations and 50 population size. For PSO-GA, we will have our $\lambda = 10$ and $\beta = 15$ as given by [21]. Our $GA_{NumMax} = 20$, $GA_{NumMin} = 10$, $GA_{MaxPS} = 25$, $GA_{MinPS} = 10$, $GA_{MinIter} = 50$, $GA_{MaxIter} = 100$.

Convergence will mean that all the members of the population are the same.
These are the results for the 3 algorithms for all 10 runs.

10.2.1 PSO

```
Trial: 1
Population Converged!
Number of Iterations: 56
Best Value: 0
Best Position: 0 0 0 0 0
Trial: 2
Population Converged!
Number of Iterations: 45
Best Value: 0
Best Position: 0 0 0 0 0
Trial: 3
Population Converged!
Number of Iterations: 48
```

```
Best Value: 0
Best Position: 0 0 0 0 0
Trial: 4
Population Converged!
Number of Iterations: 38
Best Value: 0
Best Position: 0 0 0 0 0
Trial: 5
Population Converged!
Number of Iterations: 47
Best Value: 0
Best Position: 0 0 0 0 0
Trial: 6
Population Converged!
Number of Iterations: 41
Best Value: 0
Best Position: 0 0 0 0 0
Trial: 7
Population Converged!
Number of Iterations: 52
Best Value: 0
Best Position: 0 0 0 0 0
Trial: 8
Population Converged!
Number of Iterations: 43
Best Value: 0
Best Position: 0 0 0 0 0
Trial: 9
Population Converged!
Number of Iterations: 46
Best Value: 0
Best Position: 0 0 0 0 0
Trial: 10
Population Converged!
Number of Iterations: 48
Best Value: 0
Best Position: 0 0 0 0 0
```

10.2.2 GA

```
Trial: 1
Population Converged!
Number of Iterations: 119
Best Value: 0
Best Invidividual: 0 0 0 0 0
```

```
Trial: 2
Population Converged!
Number of Iterations: 72
Best Value: 0
Best Invidividual: 0 0 0 0 0

Trial: 3
Population Converged!
Number of Iterations: 48
Best Value: 0
Best Invidividual: 0 0 0 0 0

Trial: 4
Population Converged!
Number of Iterations: 42
Best Value: 0
Best Invidividual: 0 0 0 0 0

Trial: 5
Population Converged!
Number of Iterations: 97
Best Value: 1
Best Invidividual: 0 0 0 0 1

Trial: 6
Population Converged!
Number of Iterations: 55
Best Value: 0
Best Invidividual: 0 0 0 0 0

Trial: 7
Population Converged!
Number of Iterations: 66
Best Value: 0
Best Invidividual: 0 0 0 0 0

Trial: 8
Population Converged!
Number of Iterations: 56
Best Value: 0
Best Invidividual: 0 0 0 0 0

Trial: 9
Population Converged!
```

```
Number of Iterations: 109
Best Value: 0
Best Invidividual: 0 0 0 0 0

Trial: 10
Population Converged!
Number of Iterations: 81
Best Value: 0
Best Invidividual: 0 0 0 0 0
```

10.2.3 PSO-GA

```
Trial: 1
Population Converged!
Number of Iterations: 30
Best Value: 0
Best Position: 0 0 0 0 0

Trial: 2
Population Converged!
Number of Iterations: 26
Best Value: 0
Best Position: 0 0 0 0 0

Trial: 3
Population Converged!
Number of Iterations: 28
Best Value: 0
Best Position: 0 0 0 0 0

Trial: 4
Population Converged!
Number of Iterations: 27
Best Value: 0
Best Position: 0 0 0 0 0

Trial: 5
Population Converged!
Number of Iterations: 27
Best Value: 0
Best Position: 0 0 0 0 0

Trial: 6
Population Converged!
```

```
Number of Iterations: 28
Best Value: 0
Best Position: 0 0 0 0 0

Trial: 7
Population Converged!
Number of Iterations: 27
Best Value: 0
Best Position: 0 0 0 0 0

Trial: 8
Population Converged!
Number of Iterations: 28
Best Value: 0
Best Position: 0 0 0 0 0

Trial: 9
Population Converged!
Number of Iterations: 28
Best Value: 0
Best Position: 0 0 0 0 0

Trial: 10
Population Converged!
Number of Iterations: 27
Best Value: 0
Best Position: 0 0 0 0 0
```

As you can see, out of the 10 trials, PSO converged 10/10 with 0 as its best value, GA converged 9/10 with 0 as its best value, PSO-GA converged 10/10 with 0 as its best value. It is also clear that there are less iterations for PSO-GA compared to GA and PSO. Also, PSO-GA has closer number of iterations upon convergence.

Now we will test for the effect of each individual parameter to the whole algorithm.

We will now test our 3 algorithms for benchmark functions. Benchmark functions determine whether or not our algorithms are up to standard.

10.3 Verifying PSO-GA

We will test the PSO-GA that Garg developed with non-linear constrained problems. Each problem underwent 30 independent runs which involved 30 different initial trial solutions with a randomly generated population of size $20 \times D$ for the optimization, where D is the dimension of the problem. The termination criterion has been set either limited to a maximum number of generations (500) or to the order of relative error equal to 10^6,

whichever is achieved first. The other specific parameters of algorithms are given below:

PSO settings:

Personal and social biases (c_1 and c_2) are constants that can be used to change the weighting between personal and population experience, respectively. These biases are of course made to be random each time the velocity of each member is calculated. In our experiments, $c1$ and $c2$ are both set to 1.5. Inertia weight(w) is represented by the linear expression $w = w_{max}(w_{max}w_{min})(iter/iter_{max})$. Here, $w_{max} = 0.9$ and $w_{min} = 0.4$ respectively the initial and final values of inertia weight, $iter_{max}$ represents the maximum generation number and $iter$ is used as a generation number.

GA settings:

In our experiments, we employed a real coded standard GA. Single point crossover operations were conducted at the rate of 0.85. Mutation operation restores the capability to explore unexplored spaces. Mutation rate is set to be 0.02. The other randomly control parameters for the proposed approach are taken as $GA_{MinPS} = 10$; $GA_{MinIter} = 10$; $GA_{NumMax} = 20$; $GA_{NumMin} = 1$; $\gamma = 10$; $\beta = 15$.

10.3.1 Himmelblau's Problem

The first problem is the Himmelblau's non-linear optimization problem. Proposed by Himmelblau in his book[26]. In this problem, there are five positive design variables $X = [x_1, x_2, x_3, x_4, x_5]$, six non-linear inequality constraints, and ten boundary conditions. The problem can be stated as follows:

$$
\begin{aligned}
\text{Minimize } f(X) &= 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \\
\text{s.t.} \quad & 0 \le g_1(X) \le 92 \\
& 90 \le g_2(X) \le 110 \\
& 20 \le g_3(X) \le 25 \\
\text{where } g_1(X) &= 85.334407 + 0.0056858x_2x_5 + \mathbf{0.0006262}x_1x_4 - 0.0022053x_3x_5 \\
g_2(X) &= 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 - 0.0021813x_3^2 \\
g_3(X) &= 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \\
78 \le x_1 \le 102; \quad & 33 \le x_2 \le 45; 27 \le x_3, x_4, x_5 \le 45
\end{aligned}
$$

The number in bold can also be replaced by 0.00062 which is a different formulation.

10.3.2 Design of a pressure vessel

The next problem is the engineering problem wherein a pressure vessel is designed given in figure 21

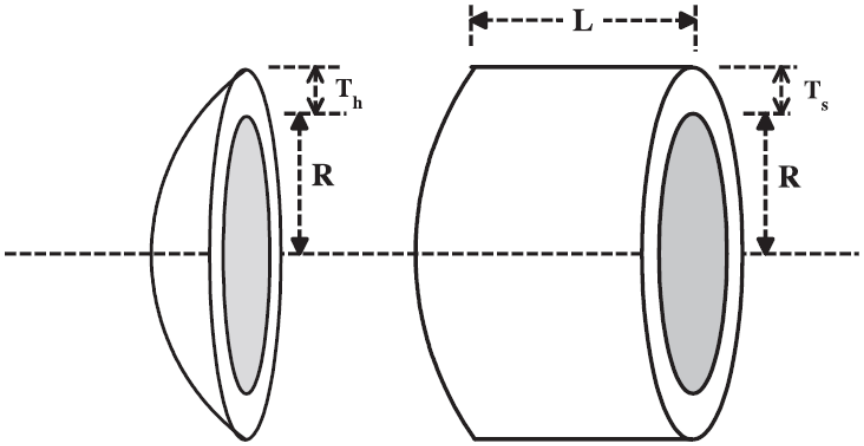

Figure 21: Design of a pressure vessel figure[21]

A cylindrical vessel capped at both ends by hemispherical heads as seen on the figure 21 needs to be designed such that it can hold a working pressure of $2000psi$ and a maximum volume of $720ft^3$ of compressed air. Using rolled steel plate, the shell is made in two halves that are joined by two longitudinal welds to form a cylinder. The objective is to minimize the total cost, including the cost of material, forming and welding[27]. There are four design variables associated with it, namely as the thickness of the pressure vessel, $T_s = x_1$, thickness of the head, $T_h = x_2$, inner radius of the vessel, $R = x_3$, and length of the vessel without heads, $L = x_4$ i.e. the variable vectors are given (in inches) by $X = (T_s, T_h, R, L) = (x_1, x_2, x_3, x_4)$. Then, the mathematical model of the problem is summarized as

$$\begin{aligned}
\text{Minimize } f(X) &= 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \\
\text{s.t. } g_1(X) &= -x_1 + 0.0193x_3 \leq 0 \\
g_2(X) &= -x_2 + 0.0095x_3 \leq 0 \\
g_3(X) &= -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0 \\
g_4(X) &= x_4 - 240 \leq 0 \\
1 \times 0.0625 \leq x_1, x_2 &\leq 99 \times 0.0625,\ 10 \leq x_3, x_4 \leq 200
\end{aligned}$$

The thickness of the shell and head are required to be of standard size (multiples of 0.625 inches - this is the thickness of regularly produced steel sheets). Both T_s and T_h are to not exceed two inches. The thickness of the shell is not to be less than 1.1 inches. The thickness of the head is not to be less than 0.6 inches. In order to seal the vessel, we must weld the caps to the cylinder.

10.3.3 Welded beam design problem

The next problem is the engineering problem wherein a welded beam is designed given in figure 22 taken from Rao[28].

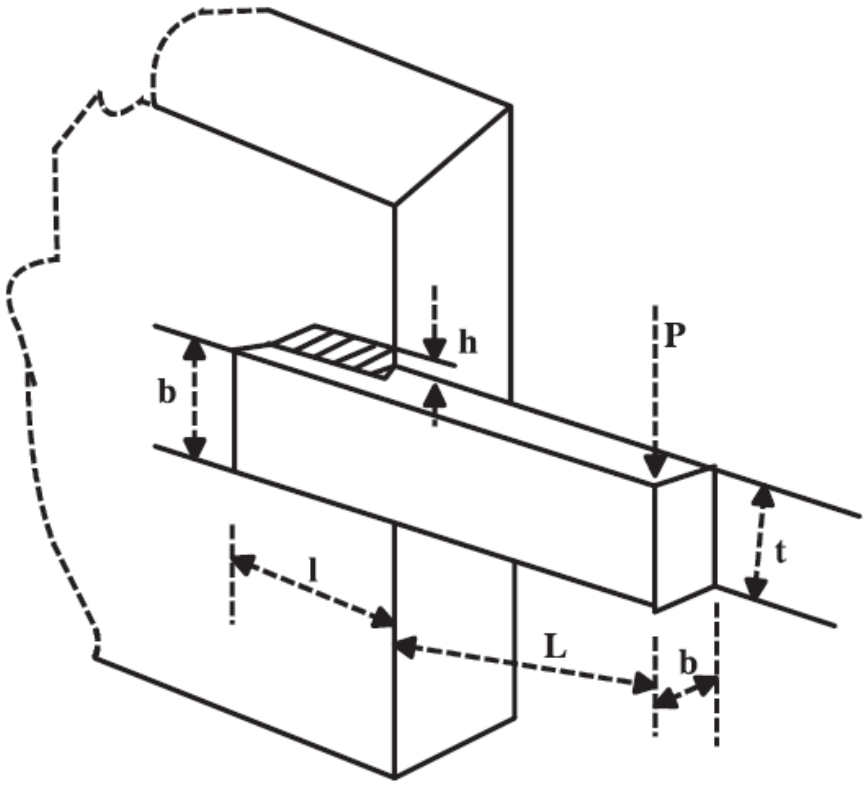

Figure 22: Welded beam design problem[21]

The objective is to find the minimum fabricating cost of the welded beam subject to constraints on shear stress (τ), bending stress in the beam (θ), buckling load on the bar (P_c), the end deflection of the beam (δ), and side constraints. There are four design variables associated with this problem, namely, thickness of the weld $h = x_1$, length of the welded joint $l = x_2$, the width of the beam $t = x_3$ and thickness of the beam $b = x_4$ i.e. the decision vector is $X = (h, l, t, b) = (x_1, x_2, x_3, x_4)$. The mathematical formulation of the objective functions $f(X)$ which is the total fabricating cost mainly comprised of the set-up, welding labor, and material cost, are as follows:

$$
\begin{aligned}
\text{Minimize } f(X) &= 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (14 + x_2) \\
\text{s.t. } g_1(X) &= \tau(X) - \tau_{\max} \leq 0 \\
g_2(X) &= \sigma(X) - \sigma_{\max} \leq 0 \\
g_3(X) &= x_1 - x_4 \leq 0 \\
g_4(X) &= 0.125 - x_1 \leq 0 \\
g_5(X) &= \delta(X) - 0.25 \leq 0 \\
g_6(X) &= P - P_c(X) \leq 0 \\
g_7(X) &= 0.10471 x_1^2 + 0.04811 x_3 x_4 (14 + x_2) - 5 \leq 0 \\
0.1 \leq x_1 \leq 2; \quad & 0.1 \leq x_2 \leq 10; \quad 0.1 \leq x_3 \leq 10; \quad 0.1 \leq x_4 \leq 2
\end{aligned}
$$

where τ is the shear stress in the weld, τ_max is the allowable shear stress of the weld ($= 13600psi$), σ the normal stress in the beam, σ_max is the allowable normal stress for the beam material ($= 30000psi$), P_c is the bar buckling load, P the load ($= 6000lb$), and δ the beam end deflection. The first constraint g_1 ensures that the maximum developed shear stress is less than the allowable shear stress of the weld material. The second constraint, g_2, checks that the maximum developed normal stress is lower than the allowed normal stress in the beam. The third constraint, g_3, checks that the beam thickness ensures exceeds that of the weld. The fourth constraint, g_4, checks that the weld thickness is above given minimum, and the constraint, $g5$, is to ensure that the end deflection of the beam is less than the predefined amount. The sixth constraint, $g6$, makes sure that the load on the beam is not greater than the allowed buckling load. The shear stress τ has two components, namely primary stress(τ_1) and secondary stress(τ_2) given as

$$\tau(X) = \sqrt{\tau_1^2 + 2\tau_1\tau_2\left(\frac{x_2}{2R}\right) + \tau_2^2} \quad ; \quad \tau_1 = \frac{P}{\sqrt{2}x_1x_2} \quad ; \quad \tau_2 = \frac{MR}{J},$$

where

$$M = P\left(L + \frac{x_2}{2}\right) \quad ; \quad J(X) = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}$$

are known as moments and polar moment of inertia respectively while the other terms associated with the model are as follows

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \quad ; \quad \sigma(X) = \frac{6PL}{x_4x_3^2} \quad ; \quad \delta(X) = \frac{6PL^3}{Ex_3^3x_4}$$

$$P_c(X) = \frac{4.013E\sqrt{\frac{x_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right)$$

$$G = 12 \times 10^6 psi, \quad E = 30 \times 10^6 psi, P = 6000lb, \quad L = 14in$$

10.3.4 Gear train design

The next problem is the unconstrained problem Gear train design. This problem has four integer variables and was introduced by Sandgren[29]. It consists of the minimization of the cost of the gear ratio of the gear train. The gear ratio is defined, for the decision variable $X = (T_d, T_b, T_a, T_f) = (x_1, x_2, x_3, x_4)$, as follows:

$$\text{Gear Train} = \frac{x_1x_2}{x_3x_4}$$

where T_i denotes the number of teeth of the gear wheel i and they are all integers vary in the range 1260. The mathematical formulation is as follows:

$$\text{Minimize } f(X) = \left(\frac{1}{6.931} - \frac{x_1 x_2}{x_3 x_4}\right)^2$$
$$12 \leq x_1, x_2, x_3, x_4 \leq 60 \quad ; \quad x_i's \in \mathbb{Z}^+$$

11 Results

11.1 Verifying PSO-GA

11.1.1 Himmelblau's Problem

The summary of results for the Himmelblau's Problem having the bold number as 0.0006262 are as follows:

```
Best OverAll Value: -30665.538566166142000
Position: 78.000000000000000 33.000000108082112 29.995256102178057
 44.999996572512927 36.775814148884123
Constraints:
G1: 92.000000000000000
G2: 94.915402310635045
G3: 20.000000006421615
Mean: -30648.963630066031000
Median: -30661.906871031875000
Standard Deviation:38.044100042155677
Worst Overall Value: -30488.180330955009000
```

The summary of results for the Himmelblau's Problem having the bold number as 0.00026 are as follows:

```
Best OverAll Value: -31025.554498975318000
Position: 78.000000000000000 33.000120765109422 27.071062580978822
44.999999999998117 44.969039277900599
Constraints:
G1: 91.999998379607803
G2: 97.207712770554437
G3: 20.000000000000480
Mean: -31014.843872873029000
Median: -31020.313167609987000
Standard Deviation:18.265991935954112
Worst Overall Value: -30932.261154305350000
```

The two tables in figure 23 shows the results that Garg[21] got in his study in which he compared it to other algorithms. As we can see, Garg ended with the result for version I is $X = [78.00, 33.00, 29.99517417, 45.00, 36.7757340]$ with the objective function value $f(x) = 30665.56614$. We got $X = [78.00, 33.00, 29.995256102178057, 44.999996572512927, 36.775814148884123]$ and the objective function value $f(x) = -30665.538566166142000$ which is less fit but close to the value by about 0.02. In version 2, Garg obtained the optimal solution $X = [78.00, 33.00, 27.07095, 45.00, 44.9691668]$ with corresponding function value 31025.57471. We got $X = [78.00, 33.000120765109422, 27.071062580978822, 44.999999999998117, 44.969039277900599]$ and the objective function value $f(x) = -31025.554498975318000$ which is more fit than the result that garg obtained. Although we got a less fit value in version 1, we did not violate any constraints. Comparing the statistical results, we have a higher deviation in values and also less similar results for 30 trials. This means that our implementation is not tuned enough.

11.1.2 Design of a pressure vessel

The summary of the results for the Design of a pressure vessel are as follows:

```
Best OverAll Value: 5880.670847119796200
Position: 0.778168641383470 0.383036377883055 40.319618724532091 199.999999993967120
Constraints:
G1: -0.000000000000000
G2: 0.000000000000000
G3: 0.000000000000000
G4: -40.000000006032877
Mean: 6083.765307149724300
Median: 6040.517399555617900
Standard Deviation:172.172683824856050
Worst Overall Value: 6385.154177201944500
```

The two tables in figure 24 shows the results that Garg[21] got in his study in which he compared it to other algorithms. Garg got $X = [0.7781686, 0.3846491, 40.3196187, 200.00000]$ with corresponding function value equal to $f(X) = 5885.33277$. We got $X = [0.778168641383470, 0.383036377883055, 40.319618724532091, 199.999999993967120]$ with the corresponding function value equal to $f(x) = 5880.670847119796200$. We actually improved the results without violating any constraints although there is a big difference in terms of the fourth constraint having a large distance from 0. If we compare the statistical results, we have a high standard deviation which means that the results we obtained for 30 runs vary in results.

11.1.3 Welded beam design problem

The summary of the results for the Welded beam design problem are as follows:

```
Best OverAll Value: 1.695247164930492
Position: 0.205729639793472 3.253120040656620 9.036623910195278 0.205729639793472
```

```
Constraints:
G1: 0.000000000000000
G2: 0.000000000000000
G3: -0.000000000000000
G4: -0.080729639793472
G5: -0.228310483876742
G6: -0.000000575920239
G7: -3.452425532400220
Mean: 1.698010492274647
Median: 1.695309853463830
Standard Deviation:0.006772856924129
Worst Overall Value: 1.726940756379066
```

The two tables in figure 25 shows the results that Garg[21] got in his study in which he compared it to other algorithms. Garg got the function value of $f(X) = 1.6952471$ corresponding to decision variable $X = [0.2057296, 3.2531200, 9.0366239, 0.2057296]$. We got $X = [0.205729639793472, 3.253120040656620, 9.036623910195278, 0.205729639793472]$ with the corresponding function value equal to $f(x) = 1.695247164930492$. We got an exact result if we truncate the remaining decimal placed but we can see that the constraint G7 is not too close to 0. Basing from the statistical results, we see the same results from the previous problems, which is that the implementation is not as efficient and as effective as that of Garg's.

11.1.4 Gear train design

The summary of the results for the Gear train design are as follows:

```
Best OverAll Value: 0.000000000002701
Position: 19.000000000000000 16.000000000000000 49.000000000000000 43.000000000000000
Gear Ratio: 0.144280968201234
Mean: 0.000000000095493
Median: 0.000000000023078
Standard Deviation:0.000000000243864
Worst Overall Value: 0.000000000992158
```

The two table in figure 26 shows the results that Garg[21] got in his study in which he compared it to other algorithms. Garg got the function value of $f(X) = 2.70085 \times 10^{12}$ while we obtained function value of $f(x) = 0.000000000002701$. Also for the gear ratio, Garg obtained 0.14428096 while we obtained 0.144280968201234. As seen by our results, we got similar outputs.

11.2 Further Experimentation

Summary of results taken from the four test problems are seen in table 1. As we can see, the best converged values may be close but there is a fault present in the results. The fault being that the output of each run was actually the global best and not the final best members.

Table 1: Summary of Results

Problem	Population Size	Converged Runs	Global Best Value
Himmelblau (6262)	100	8/30	-30665.537286550927000
Himmelblau (26)	100	7/30	-31025.554498975318000
Pressure Vessel	80	9/30	5901.554627225623300
Welded Beam	80	16/30	1.695247164981284
Gear Train	80	7/30	0.144280968201234

Therefore, we tested the process again but for different population sizes. Himmelblau's Problem version I (6262). The results are shown on table 2. As we can see, the population can only converge if the population is allowed to undergo a long generation count. It is also clear that the solutions found are better as more members are added to the population.

Table 2: Different Population Sizes and Maximum Iterations for Himmelblau's Problem version 1

Pop Size	**Max Gen**	**# of Converged**	**Best Overall**	**Mean**
250	500	0/10	n/a	-30656.59249
250	1000	2/10	-30665.13693	-30651.19292
500	500	0/10	n/a	-30660.77214
500	1000	2/10	-30665.46165	-30663.08577
1000	500	0/10	n/a	-30664.91210

Himmelblau's Problem v. II The results are shown on table 3. Similar to the first version, the population converges if it is allowed to undergo a long generation count. It is also clear that the solutions found are better as more members are added to the population. Design

Table 3: Different Population Sizes and Maximum Iterations for Himmelblau's Problem version 2

Pop Size	**Max Gen**	**# of Converged**	**Best Overall**	**Mean**
250	500	0/10	n/a	-31022.039425
250	1000	2/10	-31023.548855	-31024.977514
500	500	1/10	-31025.030580	-31025.287462
500	1000	2/10	-31025.502824	-31024.865646
1000	500	0/10	n/a	-31025.052578

of a Pressure Vessel. The results are shown on table 4. Similarly, the population converges more if it is allowed to undergo a longer generation count. There seems to be no relationship

between the number of members in the population and the solution obtained. As long as there is a large maximum generation count, the population will converge but is seems that the number of members affect how often the population converges.

Table 4: Different Population Sizes and Maximum Iterations for Pressure Vessel Design

Pop Size	**Max Gen**	**# of Converged**	**Best Overall**	**Mean**
250	500	2/10	5880.670847	5939.916628
250	1000	7/10	5880.670847	5972.922937
500	500	3/10	5880.671123	5929.322478
500	1000	2/10	5880.670847	5880.698579
1000	500	1/10	5923.720628	5938.594038

Welded Beam Problem. The results are shown on table 5. Similarly, the population converges if it is allowed to undergo a long generation count. Judging from the results of the previous problem, we see that the same relation occurs wherein the number of members in the population affects how often it converges.

Table 5: Different Population Sizes and Maximum Iterations for Welded Beam Problem

Pop Size	**Max Gen**	**# of Converged**	**Best Overall**	**Mean**
250	500	4/10	1.695247	1.695249
250	1000	6/10	1.695247	1.695384
500	500	1/10	1.695247	1.695378
500	1000	1/10	1.695247	1.695247
1000	500	0/10	n/a	1.695247

Gear Train Ratio. The results are shown on table 6. Similarly, the population converges if it is allowed to undergo a long generation count. The results are unique in that even though there is an increase in population size and maximum generation count, it rarely converges.

Table 6: Different Population Sizes and Maximum Iterations for Gear Train Ratio

Pop Size	**Max Gen**	**# of Converged**	**Best Overall**	**Mean**
250	500	1/10	0.000000000002701	0.000000000016965
250	1000	0/10	n/a	0.000000000016965
500	500	1/10	0.000000000002701	0.000000000008814
500	1000	1/10	0.000000000002701	0.000000000012890
1000	500	0/10	n/a	0.000000000008814

References

[1] R. C. Eberhart and J. Kennedy, "Particle swarm optimization," 1995.

[2] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," 1995.

[3] R. C. Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," *Evolutionary Computation*, May 2001.

[4] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 25–34, July 1987.

[5] F. Heppner and G. U.

[6] A. Kaveh and S. Talatahari, "Engineering optimization with hybrid particle swarm and ant colony opitmization," *Asian Journal of Civil Engineering*, vol. 10, pp. 611–628, 2009.

[7] M. Clerc, "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization," *Evolutionary Computation*, July 1999.

[8] M. Obitko, "Intoduction to genetic algorithms."

[9] J. Carr, "An introduction to genetic algorithms," 2014.

[10] M. Omran, "Codeq: an effective metaheuristic for continuous global optimisation," *International Journal of Metaheuristics*, vol. 1, pp. 108–131, 2010.

[11] R. Storn and K. Price, "Differential evolution a simple and efficient heuristic for global optimization over continuous spaces," *Global Optimization*, vol. 11, pp. 341–359, 1997.

[12] R. Gamperle, S. D Muller, and A. Koumoutsakos, "A parameter study for differential evolution," vol. 10, pp. 293–298, 08 2002.

[13] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," vol. 9, pp. 448–462, 2005.

[14] K. Price, R. Storn, and J. Lampinen, *Differential Evolution A Practical Approach to Global Optimization*. Springer-Verlag Berlin, 2005.

[15] H. R. Tizhoosh, "Opposition-based learning: A new scheme for machine intelligence," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, vol. 1, pp. 695–701, Nov 2005.

[16] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition-based differential evolution," *IEEE Transactions on Evolutionary Computation*, vol. 12, pp. 64–79, Feb 2008.

[17] T. Xiang, X. Liao, and K.-w. Wong, "An improved particle swarm optimization algorithm combined with piecewise linear chaotic map," vol. 190, pp. 1637–1645, 07 2007.

[18] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 58–73, Feb 2002.

[19] J. Sun, B. Feng, and W. Xu, "Particle swarm optimization with particles having quantum behavior," in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, vol. 1, pp. 325–331 Vol.1, June 2004.

[20] J. Sun, W. Xu, and B. Feng, "A global search strategy of quantum-behaved particle swarm optimization," in *IEEE Conference on Cybernetics and Intelligent Systems, 2004.*, vol. 1, pp. 111–116 vol.1, Dec 2004.

[21] H. Garg, "A hybrid pso-ga algorithm for constrained optimization problems," *Applied Mathematics and Computation*, February 2016.

[22] Z. Michalewicz and M. Schoenauer, "Evolutionary algorithms for constrained parameter optimization problems," *Evolutionary Computation*, vol. 4, pp. 1–32, 1996.

[23] Q. He and L. Wang, "An effective co-evolutionary particle swarm optimization for constrained engineering design problems," *Engineering Applications of Artificial Intelligence*, vol. 20, pp. 89–99, 2007.

[24] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, pp. 311–338, June 2000.

[25] S. He, E. Prempauin, and Q. H. Wu, "An improved particle swarm optimizer for mechanical design optimization problems," *Engineering Optimization*, vol. 36, pp. 585–605, 2004.

[26] D. M. Himmelblau, *Applied nonlinear programming.* McGraw-Hill, 1976.

[27] B. Kannan and S. Kramer, "An augmented lagrange multiplier based method for mixed integer discrete continuous optimization and its applications to mechanical design," *Journal of Mechanical Design*, vol. 116, pp. 318–320.

[28] S. Rao, *Engineering optimization: Theory and practice.* John Wiley and Sons, 1996.

[29] E. Sandgren, "Nonlinear integer and discrete programming in mechanical design," *American Society of Mechanical Engineers, Design Engineering Division (Publication) DE*, vol. 14, pp. 95–105, 1988.

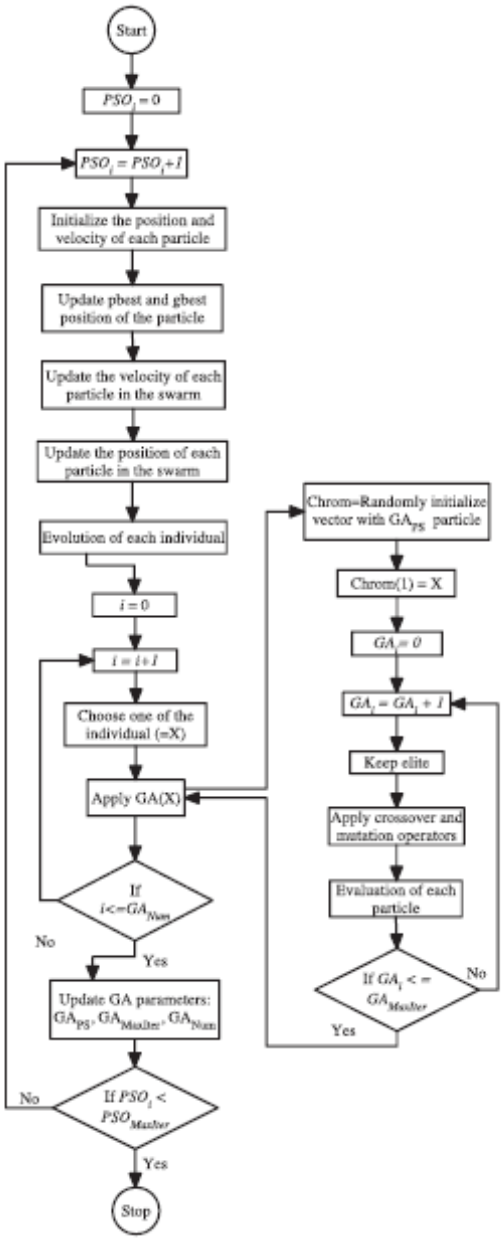

Figure 18: PSO-GA Algorithm[21]

D	number of decision variables.
M	number of constraints.
S	feasible search space.
f_w	worst feasible solution.
g_j	j th constraint function.
$f(\cdot)$	cost function.
γ	decreasing rate of no. of individuals that effected by GA.
β	increasing rate of GA maximum iteration.
GA_{PS}	population size of the particles in GA.
$GA_{MinIter}$	minimum number of iteration in GA.
$GA_{MaxIter}$	maximum number of iteration in GA.
GA_{Num}	current number of individuals that effected by GA.
GA_{NumMin}	minimum number of individuals that effected by GA.
GA_{NumMax}	maximum number of individuals that effected by GA.
GA_{MinPS}	first population size in GA
GA_{MaxPS}	last population size in GA
PSO_i	current PSO iteration.
$PSO_{MaxIter}$	maximum number of iteration in PSO.

Figure 19: Notations used in the PSO-GA Algorithm[21]

$$GA_{Num} = GA_{NumMax} - \left(\frac{PSO_i}{PSO_{MaxIter}}\right)^{\gamma} \times (GA_{NumMax} - GA_{NumMin})$$

$$GA_{PS} = GA_{MinPS} + \left(\frac{PSO_i}{PSO_{MaxIter}}\right)^{\gamma} \times (GA_{MaxPS} - GA_{MinPS})$$

$$GA_{MaxIter} = GA_{MinIter} + \left(\frac{PSO_i}{PSO_{Maxiter}}\right)^{\beta} \times (GA_{MaxIter} - GA_{MinIter})$$

Figure 20: Equations used in the PSO-GA Algorithm[21]

Table 1
Optimal results for Himmelblau's nonlinear optimization problem (NA means not available).

Version	Methods	Design variables						Constraints		
		x_1	x_2	x_3	x_4	x_5	$f(X)$	$0 \le g_1 \le 92$	$90 \le g_2 \le 110$	$20 \le g_3 \le 25$
I	Himmelblau [28]	NA	NA	NA	NA	NA	−30373.9490	NA	NA	NA
	Homaifar et al. [29]	80.39	35.07	32.05	40.33	33.34	−30005.700	91.65619	99.53690	20.02553
	Deb [25]	NA	NA	NA	NA	NA	−30665.539	NA	NA	NA
	Lee and Geem [17]	78.00	33.00	29.995	45.00	36.776	−30665.500	92.00004[a]	98.84051	19.99994[a]
	He et al. [3]	78.00	33.00	29.995256	45.00	36.7758129	−30665.539	93.28536[a]	100.40478	20.00000
	Dimopoulos [11]	78.00	33.00	29.995256	45.00	36.775813	−30665.54	92.00000	98.84050	20.00000
	Gandomi et al. [19]	78.00	33.00	29.99616	45.00	36.77605	−30665.233	91.99996	98.84067	20.0003
	Mehta and Dasgupta [21]	78.00	33.00	29.995256	45.00	36.775813	−30665.538741	NA	NA	NA
	Present study	78.00	33.00	29.9951741	45.00	36.7757340	−30665.56614	91.99999	98.84047	20.000
II	Shi and Eberhart [15]	78.00	33.00	27.07099	45.00	44.969	−31025.561	93.28533[a]	100.40473	19.99997[a]
	Coello [38]	78.5958	33.01	27.6460	45.00	45.0000	−30810.359	91.956402	100.54511	20.251919
	Coello [10]	78.0495	33.007	27.081	45.00	44.94	−31020.859	93.28381[a]	100.40786	20.00191
	Hu et al. [30]	78.00	33.00	27.070997	45.00	44.9692425	−31025.5614	92	100.404784	20
	Fesanghary et al. [18]	78.00	33.00	27.085149	45.00	44.925329	−31024.3166	93.27834[a]	100.39612	20.00000
	Omran and Salman [2]	78.00	33.00	27.0709971	45.00	44.9692425	−31025.55626	93.28536[a]	100.40478	20.00000
	Present study	78.00	33.00	27.0709505	45.00	44.9691668	−31025.57471	91.99999	100.40476	20.000

[a] violate constraints

Table 2
Statistical results for the Himmelblau's problem (NA means not available).

Version	Methods	Best	Median	Mean	Worst	Std.
I	Deb [25]	−30665.537	−30665.535	NA	−29846.654	NA
	Lee and Geem [17]	−30665.500	NA	NA	NA	NA
	He et al. [3]	−30665.539	NA	−30643.989	NA	70.043
	Dimopoulos [11]	−30665.54	NA	NA	NA	NA
	Gandomi et al. [19]	−30665.2327	NA	NA	NA	11.6231
	Mehta and Dasgupta [21]	−30665.538741	NA	NA	NA	NA
	Present study	−30665.566141	−30665.487911	−30665.398373	−30664.624696	0.24084
II	Coello [10]	−31020.859	−31017.21369	−30984.240703	−30792.407737	73.633536
	Hu et al. [30]	−31025.56142	NA	−31025.561420	NA	0
	Fesanghary et al. [18]	−31024.3166	NA	NA	NA	NA
	Omran and Salman [2]	−31025.55626	NA	−31025.556264	NA	NA
	Present study	−31025.574717	−31025.561141	−31025.557816	−31025.492054	0.01526

Figure 23: Garg's[21] Tables with Comparative Results for Himmelblau's Problem

Table 3
Comparison of the best solution for pressure vessel design problem found by different methods.

Method	Design variables				Cost
	x_1	x_2	x_3	x_4	$f(X)$
Sandgren [33]	1.125000	0.625000	47.700000	117.701000	8129.1036
Kannan and Kramer [31]	1.125000	0.625000	58.291000	43.690000	7198.0428
Deb and Gene [32]	0.937500	0.500000	48.329000	112.67900	6410.3811
Coello[10]	0.812500	0.437500	40.323900	200.000000	6288.7445
Coello and Montes [12]	0.812500	0.437500	42.097398	176.654050	6059.946
He and Wang [4]	0.812500	0.437500	42.091266	176.746500	6061.0777
Montes and Coello [34]	0.812500	0.437500	42.098087	176.640518	6059.7456
Kaveh and Talatahari [5]	0.812500	0.437500	42.103566	176.573220	6059.0925
Kaveh and Talatahari [22]	0.812500	0.437500	42.098353	176.637751	6059.7258
Zhang and Wang [39]	1.125000	0.625000	58.290000	43.6930000	7197.7000
Cagnina et al. [7]	0.812500	0.437500	42.098445	176.6365950	6059.714335
Coello [16]	0.812500	0.437500	42.098400	176.6372000	6059.7208
He et al. [3]	0.812500	0.437500	42.098445	176.6365950	6059.7143
Lee and Geem [17]	1.125000	0.625000	58.278900	43.75490000	7198.433
Montes et al. [14]	0.812500	0.437500	42.098446	176.6360470	6059.701660
Hu et al. [30]	0.812500	0.437500	42.098450	176.6366000	6059.131296
Gandomi et al. [19]	0.812500	0.437500	42.0984456	176.6365958	6059.7143348
Akay and Karaboga [40]	0.812500	0.437500	42.098446	176.636596	6059.714339
Garg [24]	0.7781977	0.3846656	40.3210545	199.9802367	5885.4032828
Present study	0.7781686	0.3846491	40.3196187	200.000000	5885.3327736

Table 4
Statistical results of different methods for pressure vessel(NA means not available).

Method	Best	Mean	Worst	Std. Dev.	Median
Sandgren [33]	8129.1036	N/A	N/A	N/A	NA
Kannan and Kramer [31]	7198.0428	N/A	N/A	N/A	NA
Deb and Gene [32]	6410.3811	N/A	N/A	N/A	NA
Coello [10]	6288.7445	6293.8432	6308.1497	7.4133	NA
Coello and Montes [12]	6059.9463	6177.2533	6469.3220	130.9297	NA
He and Wang [4]	6061.0777	6147.1332	6363.8041	86.4545	NA
Montes and Coello [34]	6059.7456	6850.0049	7332.8798	426.0000	NA
Kaveh and Talatahari [22]	6059.7258	6081.7812	6150.1289	67.2418	NA
Kaveh and Talatahari [5]	6059.0925	6075.2567	6135.3336	41.6825	NA
Gandomi et al. [19]	6059.714	6447.7360	6495.3470	502.693	NA
Cagnina et al. [7]	6059.714335	6092.0498	NA	12.1725	NA
Coello [16]	6059.7208	6440.3786	7544.4925	448.4711	6257.5943
He et al. [3]	6059.7143	6289.92881	NA	305.78	NA
Akay and Karaboga [40]	6059.714339	6245.308144	NA	205	NA
Garg [24]	5885.403282	5887.557024	5895.126804	2.745290	5886.14928
Present study	5885.332773	5885.382053	5885.486467	0.049080	5885.3812

Figure 24: Garg's[21] Tables with Comparative Results for Design of a pressure vessel

Table 5
Comparison of the best solution for welded beam found by different methods.

Method	Design variables				Cost
	x_1	x_2	x_3	x_4	$f(X)$
Coello [10]	0.208800	3.420500	8.997500	0.210000	1.748309
Coello and Montes [12]	0.205986	3.471328	9.020224	0.206480	1.728226
Hu et al. [30]	0.20573	3.47049	9.03662	0.20573	1.72485084
Hedar and Fukushima [23]	0.205644261	3.472578742	9.03662391	0.2057296	1.7250022
He and Wang [4]	0.202369	3.544214	9.048210	0.205723	1.728024
Dimopoulos [11]	0.2015	3.5620	9.041398	0.205706	1.731186
Mahdavi et al. [36]	0.20573	3.47049	9.03662	0.20573	1.7248
Montes et al. [14]	0.205730	3.470489	9.036624	0.205730	1.724852
Montes and Coello [34]	0.199742	3.612060	9.037500	0.206082	1.73730
Cagnina et al. [7]	0.205729	3.470488	9.036624	0.205729	1.724852
Fesanghary and Mahadavi [18]	0.20572	3.47060	9.03682	0.20572	1.7248
Kaveh and Talatahari[5]	0.205729	3.469875	9.036805	0.205765	1.724849
Kaveh and Talatahari [22]	0.205700	3.471131	9.036683	0.205731	1.724918
Gandomi et al. [20]	0.2015	3.562	9.0414	0.2057	1.73121
Mehta and Dasgupta [21]	0.20572885	3.47050567	9.03662392	0.20572964	1.724855
Akay and Karaboga [40]	0.205730	3.470489	9.036624	0.205730	1.724852
Garg [24]	0.20572450	3.25325369	9.03664438	0.20572999	1.69526388
Present study	0.2057296	3.2531200	9.0366239	0.2057296	1.6952471

Table 6
Statistical results of different methods for welded beam design problem (NA means not available).

Method	Best	Mean	Worst	Std. dev.	Median
Coello [10]	1.748309	1.771973	1.785835	0.011220	NA
Coello and Montes [12]	1.728226	1.792654	1.993408	0.07471	NA
Dimopoulos [11]	1.731186	NA	NA	NA	NA
He and Wang [4]	1.728024	1.748831	1.782143	0.012926	NA
Hedar and Fukushima [23]	1.7250022	1.7564428	1.8843960	0.0424175	NA
Montes et al. [14]	1.724852	1.725	NA	1E-15	NA
Montes and Coello [34]	1.737300	1.813290	1.994651	0.070500	NA
Cagnina et al. [7]	1.724852	2.0574	NA	0.2154	NA
Kaveh and Talatahari [22]	1.724918	1.729752	1.775961	0.009200	NA
Kaveh and Talatahari [5]	1.724849	1.727564	1.759522	0.008254	NA
Gandomi et al. [20]	1.7312065	1.8786560	2.3455793	0.2677989	NA
Mehta and Dasgupta [21]	1.724855	1.724865	1.72489	NA	1.724861
Akay and Karaboga [40]	1.724852	1.741913	NA	0.031	NA
Garg [24]	1.69526388	1.69530842	1.69537060	2.836238E-5	1.69530879
Present study	1.6952471	1.6952471	1.6952471	2.192×10^{-9}	1.6952471

Figure 25: Garg's[21] Tables with Comparative Results for Welded beam design problem

Table 7
Comparison of the best solution for gear train design problem found by different methods.

	Sandgren [33]	Kannan and Kramer [31]	Deb and Goyal [37]	Gandomi et al. [19]	Present study
$T_d(x_1)$	18	13	19	19	19
$T_b(x_2)$	22	15	16	16	16
$T_a(x_3)$	45	33	49	43	43
$T_f(x_4)$	60	41	43	49	49
Gear ratio	0.146667	0.144124	0.144281	0.144281	0.14428096
$f(X)$	5.712×10^{-6}	2.146×10^{-8}	2.701×10^{-12}	2.701×10^{-12}	2.70085×10^{-12}

Table 8
Statistical results of the gear train model (NA means not available).

Algorithms	Best	Median	Mean	Worst	Std. dev.
Sandgren [33]	5.712×10^{-6}	NA	NA	NA	NA
Kannan and Kramer [31]	2.146×10^{-8}	NA	NA	NA	NA
Deb and Goyal [37]	2.701×10^{-12}	NA	NA	NA	NA
Gandomi et al. [19]	2.701×10^{-12}	NA	1.9841×10^{-9}	2.3576×10^{-9}	3.5546×10^{-9}
Present study	2.70085×10^{-12}	9.9215×10^{-10}	1.2149×10^{-9}	3.2999×10^{-9}	8.7787×10^{-10}

Figure 26: Garg's[21] Tables with Comparative Results for Gear train design