1 Preliminaries

We first define some terms and notations we will be using throughout this document.

1.1 Definitions

Algorithms An **algorithm** is a sequence of unambiguous instructions for solving problems. Every step is clear and concise, no instruction should be interpreted more than one way. **Optimization** is a mathematical technique used for solving the maximum or minimum value of a function or system of equation. In a broader sense, it is a technique used to solve for the optimum solution to a particular problem. Optimality refers to obtaining the best possible form or functionality in the sense that it is more than sufficiently efficient given a set of resources. This involves meeting an expected result with high accuracy and precision such that specifications and limitations are also achieved. 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 problem. In this sense, the model must be created in such a way that we can approach it through computable means. Hence, we define the following terms. An **Objective Function** or **Fitness Function** is the mathematical equation that is modeled after the problem such that, satisfying the function will satisfy the given problem. The objective function is important because it will determine the computability and complexity of the problem as well as the approach taken, in this case, the algorithm and its implementation. Optimization problems aim to obtain the minimum and/or maximum of certain properties related to some 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 **Design Variables** as input. We say design variables because these sequence of numbers are being used to test and determine the quality of the output. The algorithm is tasked to manipulate the values of these variables in order to get the optimal solution. In tackling real world problems, design often involves a huge amount of data collection through trial-and-error. Our variables are associated to the factors which undergo changes in values during the trial-and-error processes. The data collected should give the efficiency or numerical score of the given design variables.

A **heuristic** is a technique designed for solving a problem when classical methods are too slow for finding approximate solutions or when classical methods fail to find any exact solution at all. 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.

Although time may not only be the factors that may be taken in consideration, it is the most commonly used factor in differentiating the quality of heuristic approaches. **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 fastest and most efficient way to solve the problem is considered to help in finding solutions.

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 solution to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions. We say candidate solutions because all of these individuals may give an acceptable solution but all of them may not give the best solution. Evolution of the population takes place after repeated applications of the mentioned operators. We say evolution because members of the population changes or are somewhat different as time progresses or as the population shifts from one generation to another.

Sets A **set** is a collection of well defined objects. In this document, we will talk about sets as a collection of numbers that represent objects. 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, $A = \{2, 3, 3, 2, 1, 1\}$ is equal to $A = \{1, 2, 3\}$.

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

There are two ways of declaring membership of sets, (a) **roster method** where we define all the elements included in a set by listing or enumerating all of them; and (b) **rule method (set-builder notation)** where we define all the elements included in a set using their properties. An example of the rule method is $A = \{x \text{ is a natural number}, x < 4\}$ which can also be written as $A = \{x|x \in \mathbb{N}, x < 4\}$, to be pronounced as "the set of all x, such that x is an element of the natural numbers and x is less than 4". The vertical bar ('|') is usually pronounced as "such that", and it comes between the name of the variable you're using to stand for the elements and the rule that tells you what those elements are.

Cardinality of a set is the number of unique appearances of elements in a set. Cardinality is denoted by two vertical bars ('|') separated by the set name such as '$|A|$'. That is, using our example, the cardinality of A written as $|A|$ is 3 because A has unique elements $1, 2$ and 3.

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

A set with infinite elements is called an **infinite set**, $F = ..., 1, 2, 3....$ and $|F| = \infty$. A **countable** set is a set with the same cardinality as some subset of the set of natural numbers $\mathbb{N}$. A countable set is either a **finite** set or a **countably infinite** set, nevertheless, the elements of a countable set can always be counted one at a time and, although the counting may never finish, every element of the set is associated with a

unique natural number.
A **Venn Diagram** is a visual representation of the relationships of sets.
We say that A is a **subset** of B (written as $A \subseteq B$). If all elements of A are also elements of B. If $A = \{1, 2, 3\}$ and $B = \{1, 2, 3, 4, 5\}$ then $A \subseteq B$. However if we have the set $C = \{1, 2, 3, 6\}$, $C \nsubseteq B$ because $6 \notin B$ but $A \subseteq C$. A venn diagram of the relationships of A, B and C are shown on figure 1.

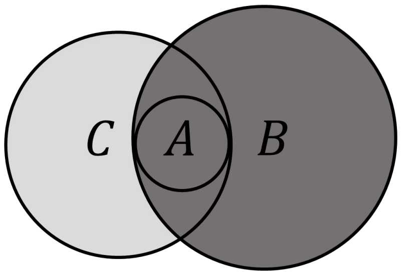


Figure 1: A Venn Diagram Showing the Relationship of A, B and C

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, $E = A \cup D = \{1, 2, 3, 4, 5, 6\}$. A venn diagram showing $A \cup D$ shaded in gray is shown on figure 1.

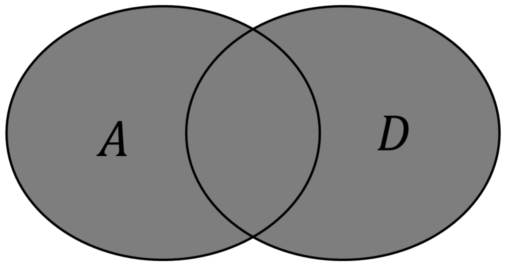


Figure 2: A Venn Diagram Showing $A \cup D$

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, $A \cap D = \{3\}$. A venn diagram of showing $A \cap D$ shaded gray is shown on figure 3.

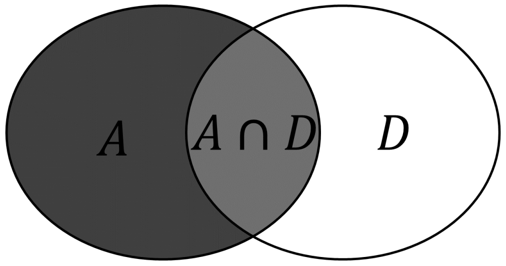


Figure 3: A Venn Diagram Showing $A \cap D$

If we have the same set A, then the **Complement** of A written as A' or $\bar{A}$ is the set containing all the elements that are not in A. That is $\bar{A} = \{x|x \in \mathbb{N}, x > 3\}$. A venn diagram of showing $\bar{A}$ shaded gray is shown on figure 4.

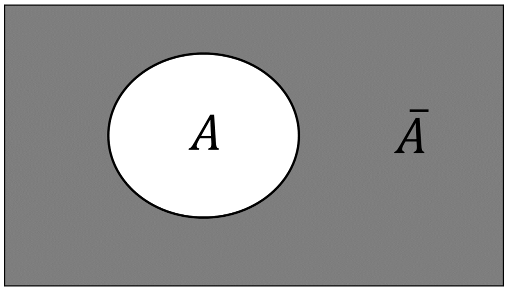


Figure 4: A Venn Diagram Showing $\bar{A}$

If we have the same sets A and D, then set **Difference (subtraction)** is defined as $A - D$ or $A\ D$ which consists of elements in A but not in D. That is, $A - D = 1, 2$. A venn diagram of showing $A - D$ shaded gray is shown on figure 5.

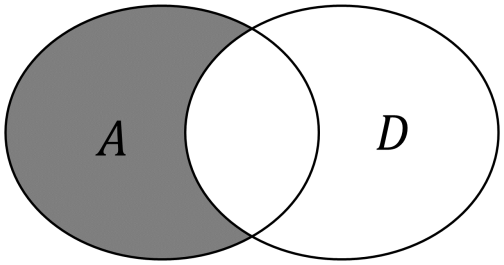


Figure 5: A Venn Diagram Showing $A - D$

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 6 .

Sequence A **sequence** is a collection of objects wherein the oder of enumeration is important (ex. a list). Unlike a set, the same elements can appear multiple times at different

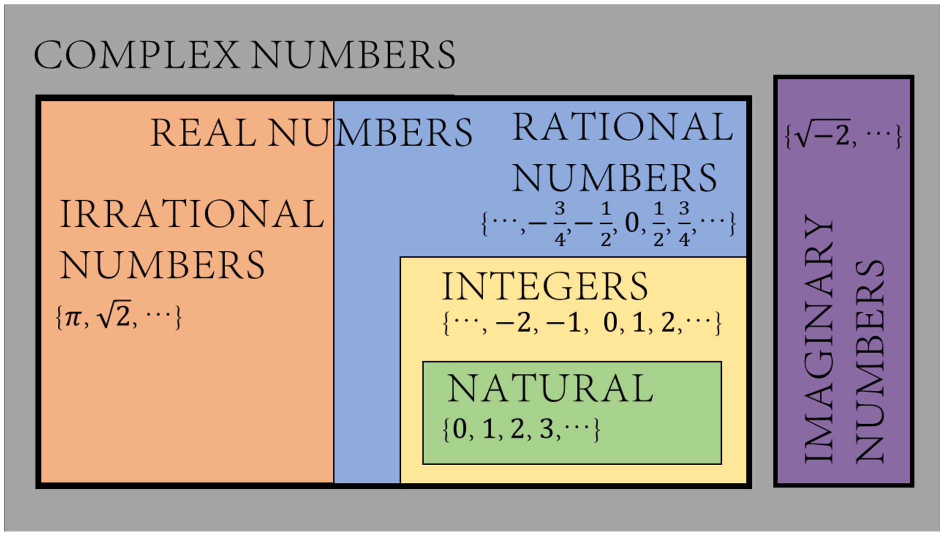

COMPLEX NUMBERS
REAL NUMBERS
RATIONAL NUMBERS
$\{\cdots, -\frac{3}{4}, -\frac{1}{2}, 0, \frac{1}{2}, \frac{3}{4}, \cdots\}$
IRRATIONAL NUMBERS
$\{\pi, \sqrt{2}, \cdots\}$
INTEGERS
$\{\cdots, -2, -1, 0, 1, 2, \cdots\}$
NATURAL
$\{0, 1, 2, 3, \cdots\}$
$\{\sqrt{-2}, \cdots\}$
IMAGINARY NUMBERS

Figure 6: A Venn Diagram Showing the Relationship of Number Sets

positions in a sequence, and order of which the elements are enumerated matters, that is if we have two sequences $(1, 2, 3)$ and $(3, 2, 1, 1)$, $(1, 2, 3) \neq (3, 2, 1, 1)$. The elements of a sequence are called **terms**. A sequence is usually denoted by parentheses ('(' and ')'), for example, the famous Fibonacci sequence is given as $(0, 1, 1, 2, 3, 5, 8, ...)$. Mathematical objects, functions or relations are usually described as sequences.
The number of elements of a sequence is called the **length** of that sequence. A sequence may be **finite** in length (ex. $(1, 2, 3, 4, 5)$) or **infinite** (ex. $(1, 2, 3, ...)$) as in sets.
Similar to sets, we can define inclusion to a sequence by generating all its elements or stating a rule. By generating all its elements, we must be sure that the sequence is finite. In case that the sequence may be infinite or has too many elements to list, then we use a rule. An example is 'the sequence of alternating 0's and 1's, starting with a 0', $(0, 1, 0, 1, 0, 1, ...)$. We can also use a formula. For example, the sequence generated by $(a_n)_{n\in\mathbb{N}} = 2n + 1$ is the sequence of odd numbers starting from 3, $(3, 5, 7, 9, ...)$.
In order to specify which element is being called, we say "**the n^{th} term**" of a sequence. For example, given the same sequence $(a_n)_{n\in\mathbb{N}} = 2n + 1$ if we want to know the 3rd element of the sequence, we write '$a_3 = 7$', we say "the third term of the sequence is the number 7".

Permutation A **permutation** is related to the act of arranging items of a set into some sequence or order. The number of all possible arrangements of a set of N items is given by $N!$. If we have the set $A = 1, 2, 3$, the permutations of set A is given as follows:

- $(1, 2, 3)$
- $(1, 3, 2)$
- $(3, 1, 2)$
- $(3, 2, 1)$
- $(2, 3, 1)$

- $(2, 1, 3)$

Geometry A **point** is a location. It has neither width nor length, even though it is visually represented as a dot for reference. Locations are usually made up of a sequence of numbers called **coordinates**. A **line** is one-dimensional, having length but no thickness. A line is composed of infinite points as it extends infinitely in both directions however, two points are enough to define a line. For example, if we are given two connected points A and B, then make-up the line $\overleftrightarrow{AB}$. A **real number line** is a line wherein each point is associated to some real number $r \in \mathbb{R}$ This makes sense because the set of real numbers is infinite. Since each point is represented as a real number, the **coordinate** of any point on the line is given by a real number. A visual representation of a real number line is shown on figure 7. As previously stated, if we want to know where a point is on the line, we simply tell what number the point represents. Hence, we also know the distance from which the point is located from our reference point, 0.

A part of a line that has defined endpoints is called a line segment. A line segment

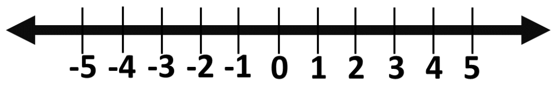


Figure 7: A Real Number Line

as the segment between A and B is written as: $\bar{AB}$. Two lines that meet at a point are called **intersecting lines**. Perpendicular lines are two line that form a 90 degree angle.

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 8. 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.

Vectors A **Vector** is a quantity having both magnitude and direction. In a coordinate plane, it is represented by an arrow as shown in figure 9. 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). A vector is mainly composed of two points in N dimensions, represented by the points on its tail and its head but it these two points are arbitrary because vectors are only concerned

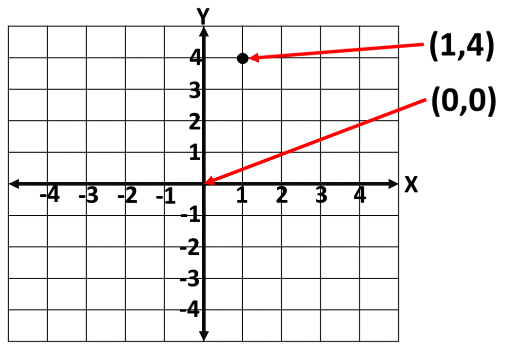
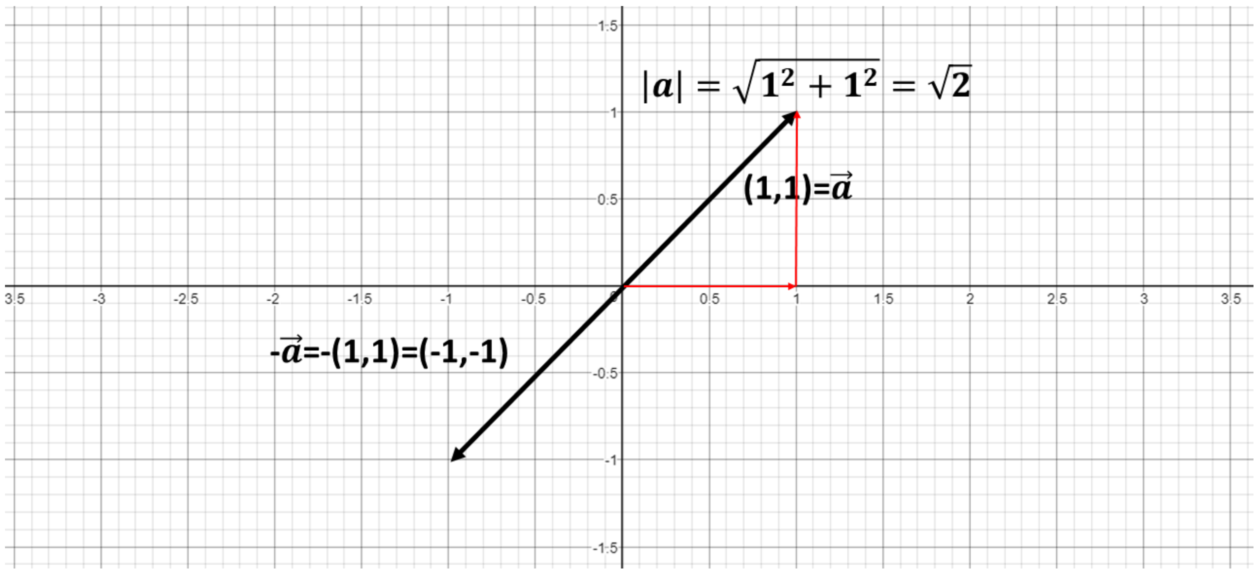


Figure 8: A Coordinate Plane

with magnitude and distance but not location. Magnitude is visually represented in length. Direction, one the other hand, is visually represented by the arrowhead. A vector is usually given in the form $< x_1, x_2, x_3, ...x_n >$ where each component x_i is the absolute numerical distance between two points in dimension $i \in (1, 2, ..., n)$. When representing vectors in two dimensions, it is broken down into two parts, x and y components. The x component is the horizontal length while the y component is the vertical length. The vector's magnitude ($|a|$) is given by the 2D Pythagorean theorem: $|a| = \sqrt{x^2 + y^2}$ where x and y are its x and y components. In higher dimensions, the same representations follow and the Pythagorean theorem for higher dimensions are used. $|a| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$ where each x_i is the component of the vector in dimension $i \in (1, 2, ..., n)$.

The number given by the Pythagorean theorem is also known as the **Euclidean**



$|a| = \sqrt{1^2 + 1^2} = \sqrt{2}$
$(1,1)=\vec{a}$
$-\vec{a}=-(1,1)=(-1,-1)$

Figure 9: Vectors in a Coordinate Plane

distance from two points, $(x, 0)$ and $(0, y)$. Euclidean distance is the length of the shortest possible path through space between two points that could be taken if there were no obstacles in between them.

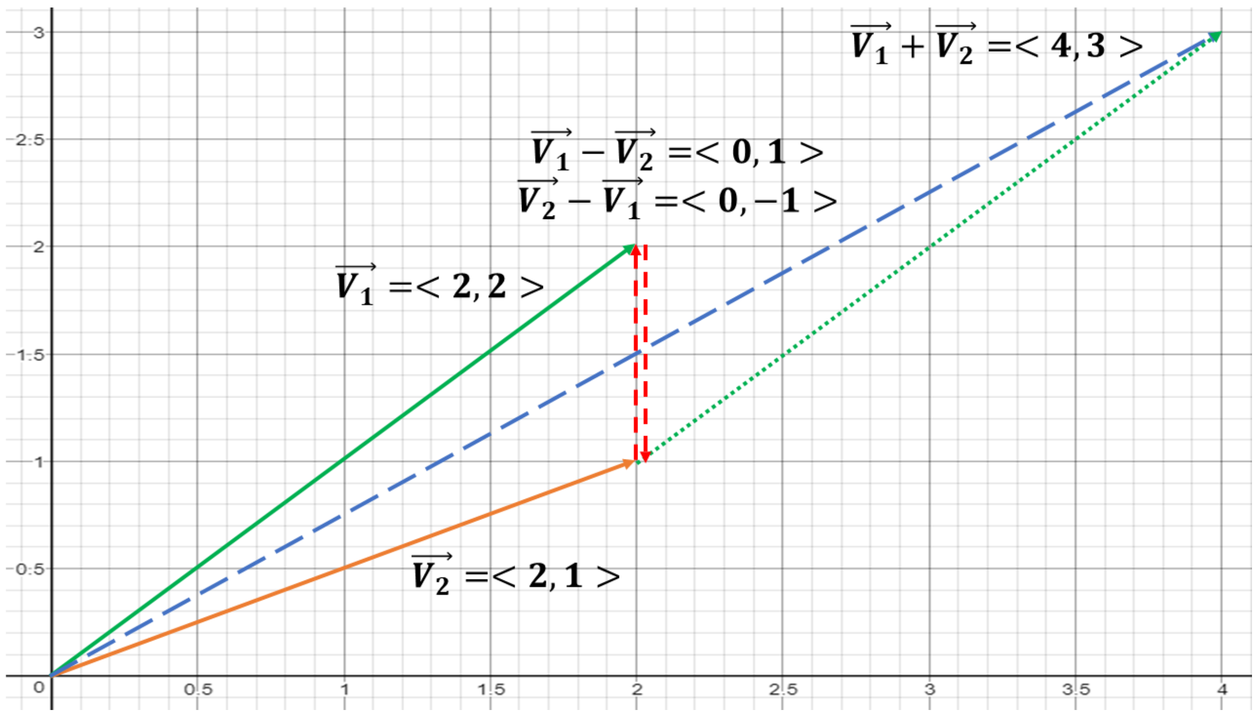


Figure 10: Adding and Subtracting Vectors (Coordinate Plane)

The negative of a vector is simply a vector having the same magnitude but of opposite direction as seen on figure 9. 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 10. 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 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.

Arrray 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 11. 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

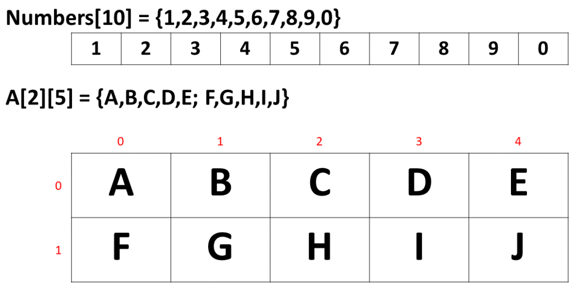

Numbers[10] = {1,2,3,4,5,6,7,8,9,0}
1
2
3
4
5
6
7
8
9
0
A[2][5] = {A,B,C,D,E; F,G,H,I,J}
0
1
2
3
4
0
A
B
C
D
E
1
F
G
H
I
J

Figure 11: Visual examples of arrays

starts from 0. In figure 11 the red numbers indicate indices of the elements. For example, if we want to access the first element in A from figure 11, 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.

Graph Theory A **graph** G is a mathematical object composed of two sets, a finite set V called the **vertices** and another set E whose elements are pairs of vertices called **edges**, expressed as $G = (V, E)$. A **vertex**, also called a **node**, is the fundamental unit needed to construct graphs. They are visually represented as points in some space S having N dimensions. In this document, they are used to represent real world objects. Later, we will assign numbers to these points to achieve discreteness, (to know what they are and what they are not). The relationships of these objects (vertices) are visually shown through lines that connect them called edges. Edges usually connect two vertices, they represent and show the relationship that exists (or not) between these vertices.

If two vertices $u, v \in V$ are connected by some edge $(u, v) \in E$, and if the edge $(v, u) \in E$ is the same edge, then we say that vertices u and v are **adjacent** and are connected by the **undirected edge** (u, v) (or (v, u)). A graph G is called an **undirected graph** if and only if it is made up of undirected edges.

However, if edge $(u, v) \in E$, and $(v, u) \in E$ are not the same edges, then we say that (u, v) is a **directed edge** from vertex u (called the edge's 'tail') to vertex v (called the edge's 'head'). If edge $(u, v) \in E$ but $(v, u) \notin E$, then we say that vertex u is **adjacent** to vertex v but vertex v is not adjacent to vertex u. A graph G is called a

directed graph if and only if it is made up of directed edges.
A graph with which every pair of vertices $u, v \in V$ is connected by an edge $(u, v) \in E$ is called a **complete graph**, denoted as $K_{|V|}$. That is, there exists an edge (u, v) in set E for any pair of u and v in set V (expressed as $\exists(u, v) \in E \ \forall u, v \in V$).
A graph is said to be a **weighted graph** if numbers are assigned to it's edges. These numbers are called **weights** or **costs**.
A **path** from vertex u to vertex v of a graph is defined as a sequence of adjacent vertices (connected by edges) that start from u and end with v. If all vertices of a path are distinct, then the path is said to be **simple**.
The **length** of a path is the total number of edges in the path.
A **directed path** is a sequence of vertices in which every consecutive pair of the vertices u and v is connected by a directed edge from u to v.
A graph is said to be connected if for every pair of vertices u and v in set V, there exists a path from u to v.
A **cycle** is a path of positive length (at least one edge) that starts and ends at the same vertex and does not traverse the same edge more than once. A graph with no cycles is said to be **acyclic**.

Shortest Path The shortest path problem is the problem of finding a path between two vertices (or nodes) u and v in a graph G such that (a) if G is unweighted, the total length is the path is minimized; (b) if G is weighted, the sum of the weights of the edges in the path is minimized.
The well known algorithms used to solve the shortest path problem are as follows:

(a) **Dijkstra's Algorithm** which solves the shortest path problem with non-negative weights. It is a single-source shortest path problem, which means that it solves the shortest path from any node $u \in V$ to any other node $v \in V$.

(b) **Floyd-Warshall Algorithm** which solves the shortest path for any two node u and v in V.

An application of this algorithm involves finding a sequence of road segments that take a vehicle from a source to a destination using a graph that represents a road network. In this representation, we can let vertices be the source, destination, intersections, land marks, etc. whatever objects that can help split the entire road systems into road segments. We then let edges be the road segments between any two vertices. We assign the costs/weights to the edges based on some information that can help us understand and/or distinguish edges that are favorable to traverse. These costs may be quantified as actual distances, cost of fuel, average amount of travel time, traffic gradient, risks involved (bridge instabilities, accident proneness, etc.) and much more depending on the realism of the model and/or data availability. The model for the amount of cost can be as simple as minimizing the amount of distance traveled or as complex as maximizing the total amount of money gained after subtracting the total money expended on fuel (affected by both distance and time), car maintenance, driver salary, etc.

TSP The **Traveling Salesman Problem (TSP)** is a problem involving generating a

Hamiltonian Cycle from a graph G. A hamiltonian path is a path in a graph which contains each vertex of the graph exactly once. A hamiltonian cycle is a hamiltonian path that starts and ends at the same vertex. The problem description are as follows:

(a) A salesman needs to visit every city (represented by vertices)

(b) He/she does not care about the order of visiting each city. As long as he/she visits each one.

(c) He/she must start and finish at the same city

(d) Each city is connected to other close by cities, or nodes, by airplanes, or by road or railway. Hence, each of the connections between the cities has one or more weights (or the cost) attached depending on the availability of transportation means.

(e) The cost describes how "expensive" it is to traverse this edge on the graph, and may be given, for example, by the cost of an airplane ticket or train ticket, or perhaps by the length of the edge, or time required to complete the traversal.

(f) The salesman wants to keep both the travel costs, as well as the distance he travels to a minimum.

The aim is to generate a path or a sequence of nodes that lets the salesman pass through all cities at most once before returning to starting city and spends the minimum amount of travel expenses and distance.
The problem is mostly concerned about generating the best arrangement of N cities among $(N-1)!$ permutations. As we can see, the amount of permutations rapidly increases as the number of cities is increased. $N-1$ because we always start with the same given city.

VRP The **Vehicle Routing Problem** is a generalization of the Traveling Salesman Problem. VRP is a problem that involves generating the best set of routes for a fleet of vehicles to service all customers in a graph. Here, there are more 'salesmen' (changed into 'vehicles' for formalities when we consider modern delivery services). VRP is concerned with delivering or collecting 'goods' to and/or from customers using a number of vehicles. A **route** is a hamiltonian cycle which starts and ends at a depot. A **depot** is where vehicles are stored or parked when they are not in use. The usual way customers and road networks are set-up is to let vertices represent the depot and customers and let the edges represent road segments that connect the vertices. Another way is to represent clusters or customers as an edge, and let the vertices serve as road intersections. This is simple but it is too simple that it does not capture individuality of customers. Hence, the former is commonly used since most adaptable models are complex.
VRP is defined on a complete undirected graph $G = (V, E)$. The set of vertices $V = 0, 1, 2, ..., n$ where each vertex $u \in V - \{0\}$ represents a customer having a non-negative demand q_u. The demand is usually the amount of goods (in some quantity) to be delivered or collected by the vehicle. The amount of goods can be measured in mass, weight, quantity, volume, bulk, etc. Vertex 0 is usually designated as the depot. Each edge $e \in E = (u, v)|u, v \in V$ is associated with a travel cost c_e or $c_{u,v}$. Travel

cost may be in terms of distance (actual, euclidean, circular, manhattan, chessboard), time (travel time, time waiting in traffic), fuel cost (convert distance and time into amount of fuel and convert that number into how much money fuel costs), monetary cost (adding up expenses, salaries, penalties) etc. There are a total of k available vehicles in the depot. The vehicles are assumed to be homogeneous and all have the same carrying capacity Q. Carrying capacity refers to the maximum amount of goods that can be carried by a vehicle at any phase or time during it's traversal of the route. The task is to develop k routes whose total travel cost is minimized such that

- Each customer is visited exactly once by a route
- Each route starts and ends at the depot
- The total demand of customers served by a route does not exceed the vehicle capacity Q
- The length of the route does not exceed a preset limit L

The last item ensures that all the drivers have the same workload.
If we consider a directed graph, then we need only to change the edges and must produce directed cycles.

2 Introduction

Waste collection industry is one of the key sectors in urban communities around the world. Today, everyone has waste to dispose due to the current lifestyle of fast food, online shopping and shipping, and non-bulk packaging. Every day, about 35,000 tons of municipal solid waste are produced in the Philippines. In Baguio City, the amount of garbage produced daily increased by more than 20% from 2008 to 2013.[1] As of 2018, Baguio City has 14 functioning waste collection trucks with a pending purchase of 4 more vehicles.[2] These trucks are responsible for servicing the 128 barangays (villages) from 3 a.m. in the morning to 11 p.m. at night. On the other hand, two compactors are utilized for waste collection in the central business district area. This move of purchasing vehicles was said to boost efficiency and to keep-up with the growing tourist influx during the weekends, holidays and the incoming summer vacation. The city conducted a Waste Analysis and Characterization Survey (WACS) where the output of garbage in the city was investigated. It was found that the residents of the city produce 402 tons of mixed waste (biodegradable and non-biodegradable) daily with 150 to 167 tons being residual waste while the rest are classified as biodegradable and recyclable. From 2017, the city has approved and is en route to establishing and developing several waste collection facilities for the next ten years, namely, centralized materials recovery facility, engineered sanitary land-fill, anaerobic digester, waste-to-energy plant, Environmental Recycling System machines, health and medical waste treatment plant, and special waste treatment plant.[3] Indeed, the city is doing it's part to reduce the carbon footprint and employ better waste management by employing the no plastic policy or the "Plastic and Styrofoam-Free Baguio Ordinance" of 2017. This city ordinance regulates the sale, distribution and use of plastic bags and styrofoam in the city. Instead of plastics and styrofoam containers, vendors are encouraged "to provide or make available to customers for free or for a cost, paper bags or reusable bags or containers made of paper or materials which are biodegradable, for the purpose of carrying out goods or other items from the point of sale.[4]

In line with these city policies and activities, we study the current efficiency of the routing and scheduling of waste collection vehicles. Specifically, we study the most efficient set of routes for the minimum amount of vehicles required to handle the job.

Vehicle routing problem is a combinatorial optimization programming problem aiming to provide service to a number of customers with a fleet of vehicles. The first article about vehicle routing was done by Dantzig and Ramser[5] in 1959. In the recent decades, vehicle routing has been the focus of some researchers and application developers (such as WasteRoute[6]). From public transportation to product deliveries, vehicle routing solutions has helped not only businesses but also the common man. Optimization of the vehicle routing problem has given rise to efficient fuel consumption as well as time management. In this study, we will focus on routing waste collection vehicles in Baguio City given time windows for servicing customers.

There are a number of ways to solve a vehicle routing problem. Researchers have used both exact algorithms and heuristic algorithms. Exact algorithms employ mathematical properties, definitions and theorems in order to logically solve a problem with a number of computations and manipulations. These kinds of approaches are able to obtain exact solutions but are too computationally complex to perform when we the problem size increases. Some methods under the exact algorithm are the Branch-and-bound method, Dynamic Programming[7], and Integer Programming. Heuristic algorithms on the other hand are able to obtain solutions to the problem but they may return sub-optimal ones. Some methods under the heuristic algorithms include Genetic Algorithms[8], Particle Swarm Optimizations[9, 10], Neighborhood Search Heuristic[11] and some hybrid algorithms[12, 13, 14]. In this study, the problem will be tackled using a hybrid metaheuristic algorithm, Particle Swarm Optimization (PSO) coupled with Genetic Algorithm (GA).

3 Vehicle Routing Problem with Time Window

The vehicle routing problem (VRP) is an non-deterministic polynomial-time hard problem. This means that finding solutions to these kinds of problems will take exponentially longer to obtain as the problem size increases. Solutions to the VRP, although difficult to produce, are quickly verifiable due to them being done in polynomial time for a deterministic Turing Machine to check. We may say that a solution is feasible but it may not be the best one (optimal) in the whole solution space. A solution may not even exist at all. The VRP is typically a combinatorial problem that deals with arranging multiple specified locations along a single path while satisfying known constraints. VRP is usually concerned with the minimization of the temporal and/or geographical aspects of traveling along paths while accommodating the most amount of customer demands along the way. Customers are usually distributed at different locations and each customer has a certain amount of demand assigned that utilizes the capacity of the vehicles servicing them. VRP models also include minimizing the number of vehicles needed to satisfy the total customer demand. In waste collection, a regular VRP can be used to model the residential sector since trucks only need to traverse along each location, collecting waste and moving them to disposal sites.

The vehicle routing problem with time windows (VRTPW) is an extension of the VRP problem. VRPTW is also an np-hard problem and given a fixed number of vehicles, it becomes an np-complete problem. Here, some customers identify a time interval when the service is needed or expected. This adds time dimension to the problem, hence, we now consider minimizing both spatial and temporal costs. Service time is also taken into consideration, this is the amount of time required for the loading and unloading processes at locations. VRPTW specifically covers the business sector in waste collection. Usually, businesses and waste collection services draft contracts for accommodation at a particular time and day.

Further extending the VRPTW, Buhrkal[11] add a lunch break, required by law, for the drivers. In addition to the previously mentioned constraints, drivers will have to take a certain amount of time to rest and eat in between any two locations at most once along the entire route.

Adding some specifications, waste collection also involves the holding capacity of a garbage truck as well as a disposal period wherein the vehicle needs to visit a disposal sight to refresh it's capacity after it is filled along the route and before the vehicle returns to the depot to retire for the day.

4 Review of Related Literature

In 1987, Solomon[15] proposed and implemented some insertion heuristics to some problems sets he created (called ”Solomon Benchmark Problems”) for benchmarking VRPTW solution methods. The first is the savings heuristics which starts out with each customer having dedicated routes, then the algorithm tries to combine the best pair of routes with each other until the minimum amount of routes are produced. The best pair of routes is decided by some savings equation which gives the amount of cost saved if two routes are combined rather than separate. The next heuristic is a greedy approach, the time-oriented nearest-neighbor heuristic which starts by selecting the ”closest” (in terms of travel distance and time) node from the depot and attaching it to the current route. We repeat the process for the last added node until the schedule is full. We then proceed to create the next route. The next heuristic introduced is the insertion heuristic which can construct routes sequentially. Each node is inserted in the best location, between two nodes in the route. The best location for insertion is determined by some function that shows how efficient the route becomes after insertion. There are three proposed ways of evaluating the efficiency of the routes each called the I1, I2, and I3 respectively. When the node is inserted, the nodes succeeding it are pushed forward, meaning that servicing these nodes are adjusted based on how much time and distance is used to accommodate the inserted node. The last heuristic discussed is the time-oriented sweep heuristic which groups customers and assigns them to a vehicle. The schedule for each vehicle is constructed from there. The results show that among the proposed heuristics, the insertion algorithm (specifically the I1) proved to be the most effective is solving the benchmark test cases because it focuses more on correct sequencing customers rather than grouping and assigning customers to vehicles.

In 2000, Son[9] utilized a Chaotic Particle Swarm Optimization (CPSO) algorithm to generate routes and schedules of the different waste collection vehicles at Danang City Vietnam. The CPSO obtained data on the roads and waste collection facilities from a Geographic Information System (GIS) that simulates a continuous environment and models the road network and waste collection system of Danag City. There are three different kinds of vehicles available, namely, tricycles, hook-lifts and forklifts which take up different roles in the waste collection system. The objective in this case was to maximize the amount of garbage collected in the simulation.

In 2005, Nuortio et.al.[16] improved the inflexible and inefficient waste collection scheduling and routing in Eastern Finland by creating a conceptual model based on the available data and employing a heuristic based on the guided variable neighborhood thresholding meta heuristic. The results showed that the schedule produced by the heuristic significantly reduced travel costs of vehicles.

In 2007, Cordeau et.al.[17] compiled and defined general models of the vehicle routing problem and it's extensions. They also compiled and cited the approaches done by researchers over the years to tackle the VRP and it's extensions. They also give a brief summary of the process of how each algorithm solves the problems presented.

In 2012, Burhkal et.al.[11] set-up a model for waste collection VRPTW with lunch breaks after two test cases, namely the Waste Management Inc. which is responsible for waste collection in parts of Northern America and the Henrik Tofteng Company responsible for handling waste collection at Denmark. These two cases have different policies for lunch break hours, limits on the number of customers served per route, and total amount collected at each route. They provided both cases with solutions using an adaptive large neighborhood search heuristic.

In 2015, Akhtar, Hannan and Basri[10] proposed a method of solving Waste Collection Vehicle Routing Problem by distributing customers into bins and creating a traveling salesman problem for each bin then applying Particle Swarm Optimization to create routes.

5 PSO

Particle Swarm Optimization (PSO) is an optimization algorithm based on a simplified avian social model. PSO was proposed by Kennedy and Eberhart on 1995. [18, 19] Their original algorithm is shown below. PSO was discovered from the 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.

1. Generate an initial population of N members composed of random positions and velocities with d dimensions from the problem space.

2. For each particle x_{id} in the population, evaluate the fitness function values $F(x_{id})$, $i \in (1, 2, 3, ..., N)$

3. Compare the fitness value with the current particle's best value (pbest). (That is, *pbest* compared to $F(x_{id})$)

   If the current fitness value is better than pbest ($pbest > F(x_{id})$), then set the pbest value equal to the fitness value ($pbest \leftarrow F(x_{id})$) as well as the pbest location to the current location of the particle in d-dimensional space ($pbest_{id} \leftarrow x_{id}$).

   else retain the pbest value and location.

4. Compare fitness value with the population's global best value (gbest). (That is, *gbest* compared to $F(x_{bd})$, b is index of the best particle in the population)

   If the current overall best fitness value is better than gbest ($gbest > F(x_{bd})$), then set the gbest value equal to the overall best fitness value ($gbest \leftarrow F(x_{Bd})$) as well as the gbest location to the current location of the overall best particle in d-dimensional space ($pbest_{gd}, g \leftarrow b$, where $gwasthepreviousbestlocationindex$).

   else retain the gbest value and location.

5. Change the velocity and position of the particles according to the equations:

$$v_{id} = v_{id} + c_1 * rand() * (pbest_{id} - x_{id}) + c_2 * rand() * (pbest_{gd} - x_{id})$$

$$x_{id} = x_{id} + v_{id}$$

6. Loop the process to step 2 until one of the following conditions are met, a sufficiently good fitness is reached or a maximum number of iterations (generations) are reached.

The original algorithm is quite simple. The population is initialized by randomly obtaining some particles within the search space and generating random velocities that are paired to each particle. There are N particles in the population. Each particle's position (x_{id}) and velocity (v_{id})is composed of d numbers where d is the dimension of the search space and $i \in (1, 2, 3, ..., N)$. We take note that each dimension of the search space is usually bounded or are in intervals $[a_j, b_j]$, $j \in (1, 2, 3, ..., d)$. a_j is the lowest number that each x_ij can be while b_j is the highest number that each x_ij can be.

Each particle's personal best value and location are recorded as *pbest* and $pbest_{id}$. At each loop, the pbest value is compared to the particle's fitness value and updated. This acts as a memory of where the particle was last at its best. Another pair of value and location are recoded which are the overall best particle's fitness value and location. The overall best particle is the particle in the current population that has currently the best fitness value. (Best is usually determined as the lowest or highest fitness value depending on the implementation) These values are known as *gbest* and $pbest_{gd}$. This pair serves as a memory of where the most optimum location is currently at. These recorded values will serve to guide each member to the most optimum location on the search space as seen on the equations at step 5.

The particle's velocity and location are changes in step 5. As we can see, there are many variables involved in the equations. They will be discussed in the next sections.

5.1 Background

We first discuss the concepts where the 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[20], Heppner and Grenander [21] 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[21], 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 R^N.

The velocity equation underwent some changes until it became:

$$V[i][d] = c1 * rand() * (pbest[i][d] - present[i][d]) + c2 * rand() * pbest[gbest][d] - present[i][d]) \tag{1}$$

where $v[i][d]$ is the d^{th} velocity component of particle i in D dimensions, $rand()$ are the randomly generated stochastic variables, $pbest[i][d]$ is the d^{th} component of the particle's best position in D dimensions, $pbest[gbest][d]$ is the d^{th} component of the population's best particle's position ($gbest$) in D dimensions, $present[i][d]$ is the d^{th} component of the particle's current position in D dimensions, $c1$ and $c2$ are constant numbers, $x \in (1, 2, 3, ..., n)$

Eberhart and Kennedy [18] 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 commit it activities along 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.

5.2 Further Developments

Eberhart and Shi[22] explains that the terms of the velocity vector seen on step 5 of the original algorithm are all important. The first term ($v_i d$) 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 locations $pbest_{id}$ and $pbest_{gd}$. In addition, without it, the search space will shrink and never grow since it will only move toward the centroid of it's recorded locations $pbest_{id}$ and $pbest_{gd}$. The two terms $c_1 * rand() * (pbest_{id} - x_{id})$ and $c_2 * rand() * (pbest_{gd} - x_{id})$ concerning the personal best and global best comparison with the current position is necessary to keep the particles from flying in the same direction for every iteration and leaving the search space.

Eberhart and Shi[22] further improved the original algorithm proposed by Eberhart and Kennedy[18] by introducing inertia weight. Inertia weight is responsible for balancing global and local exploration. The new velocity equation becomes

$$v_{id} = v_{id} * w_{id} + c_1 * rand() * (pbest_{id} - x_{id}) + c_2 * rand() * (pbest_{gd} - x_{id}) \qquad (2)$$

where the new variable w_{id} is the inertia weight. Eberhart and Shi[22] 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.

5.2.1 Fixing Convergence

Although PSO is simple in implementation and design, it had certain flaws. It has high computational costs which is given by it slow convergence.[23] 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[24]. The formula then becomes

$$v_{id} = K[v_{id} + c_1 * rand() * (pbest_{id} - x_{id}) + c_2 * rand() * (pbest_{gd} - x_{id})]$$

where $K = \frac{2}{|2-\varphi-\sqrt{\varphi^2-4}|}$, $\varphi = c_1 + c_2$ and $\varphi > 4$.

5.2.2 Chaos Search

Chaos is a characteristic of non-liner system that includes infinite unstable periodic motions and depends on initial conditions.[25] Due to its uncertainty and stochastic properties, chaotic sequences have been used to replace random generated numbers and to enhance the performance of heuristic optimization algorithms such as GA, PSO and others. There are several chaotic maps available with different properties and 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 [26],

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

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 [26] 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 [27]:

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

Although it is not very different from the equation of Kennedy et.al.[18], 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.

5.2.3 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[28]. Sun et.al.[28] 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.[28, 29]
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.

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

6 GA

Genetic Algorithm (GA) is an evolutionary algorithm developed by John Holland et. al.[30] 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[30] 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 6.2

1. Generate random population of N chromosomes in the solution space of D dimensions.
2. Evaluate the fitness $F(x)$ of each chromosome x_{id} in the population where $i \in (1, 2, 3, ..., N)$ and $d = (1, 2, 3, ..., D)$
3. Create a new population by repeating the following steps until the new population is complete
   (a) (Selection) Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chances of selection)
   (b) (Crossover) With a crossover probability, perform crossing for selected parents to produce new offspring (children). If no crossover was performed, offspring is an exact copy of parents. (Implementor may choose to select only 1 child to be placed in the population through some selection means)
   (c) (Mutation) With a mutation probability, mutate new offspring at some positions $x_d, d \in (1, 2, 3, ..., D)$
   (d) (Accepting) Place new offspring in a new population
4. Replace the old population with the new population for further iterations of GA
5. Loop the process to step 2 until one of the following conditions are met, a sufficiently good fitness is reached or a maximum number of iterations (generations) are reached.

We now discuss what happens at each part of the algorithm.

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 not be able obtain diversity during crossover hence, only a small part of the search space is explored depending on the values of the initial population. On the other hand, if there are too many chromosomes, GA slows down and many of the elements of the initial population tend to be repeated, hence overestimation occurs. After several years of research, it was determined that after some limit (which depends mainly on the encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster.[31] This is because the population size becomes too big for the solution space, or the number of computations needed becomes too large and redundant.

6.2.2 Fitness Evaluation

Next is to evaluate the fitness function $f(x)$ for each chromosome x_i in the generation. A fitness function is the function that the algorithm is trying to optimize. The word "fitness" is taken from the evolutionary theory. It tests and quantifies how 'fit' each potential solution is with respect to the problem.[32] 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 12. 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 13.

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 14.

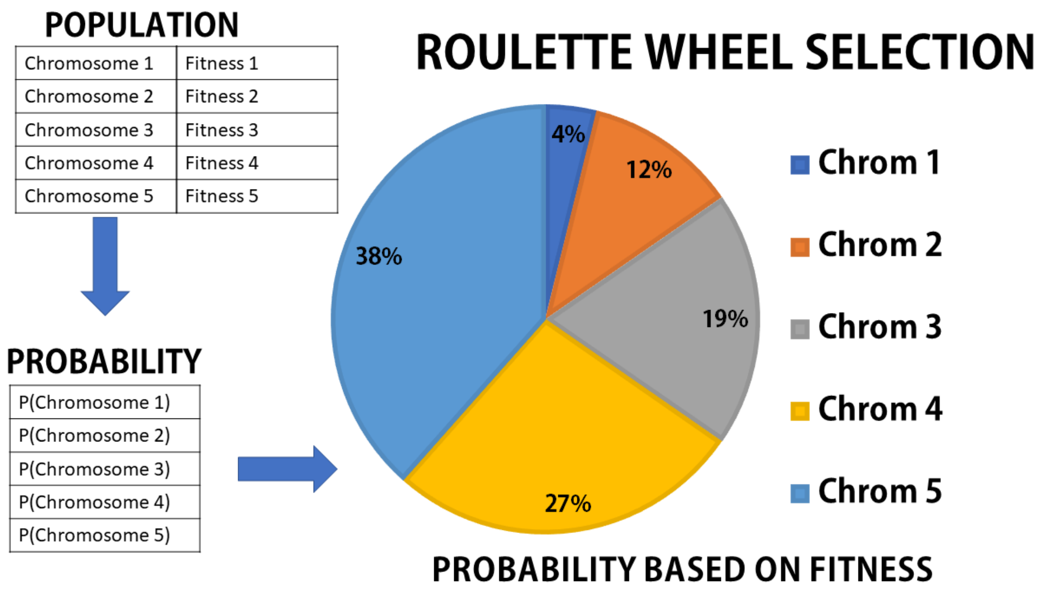


Figure 12: Roulette Wheel Selection

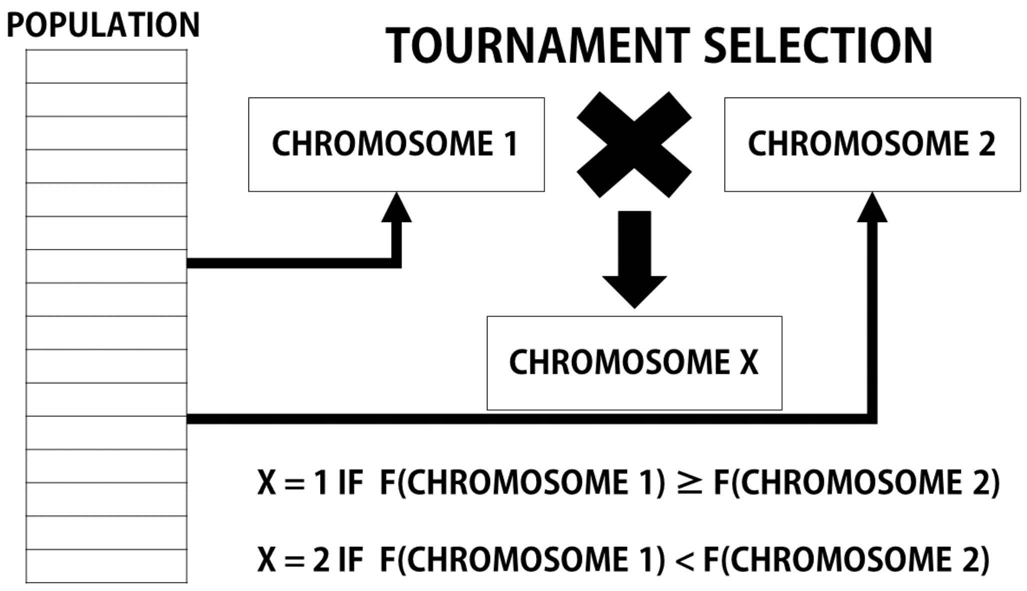

POPULATION
TOURNAMENT SELECTION
CHROMOSOME 1
CHROMOSOME 2
CHROMOSOME X
X = 1 IF F(CHROMOSOME 1) ≥ F(CHROMOSOME 2)
X = 2 IF F(CHROMOSOME 1) < F(CHROMOSOME 2)

Figure 13: Simple Elimination Selection

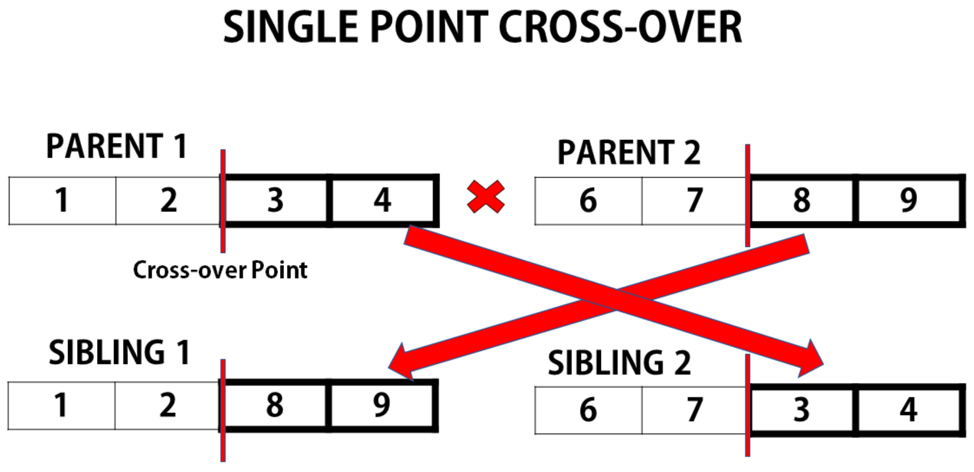


Figure 14: Singe Point Cross-Over

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 15. 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.

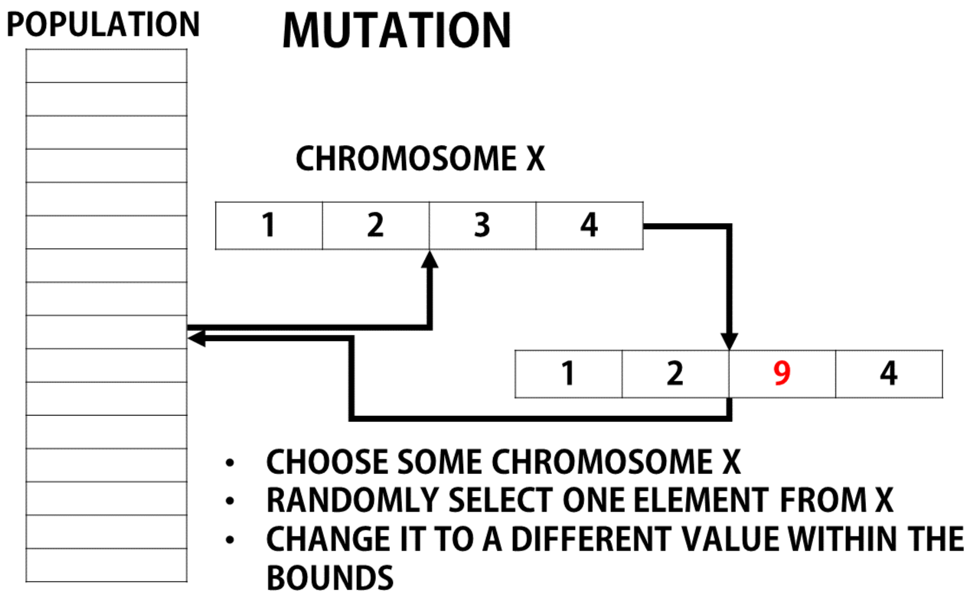

POPULATION
MUTATION
CHROMOSOME X
1
2
3
4
1
2
9
4
• CHOOSE SOME CHROMOSOME X
• RANDOMLY SELECT ONE ELEMENT FROM X
• CHANGE IT TO A DIFFERENT VALUE WITHIN THE BOUNDS

Figure 15: 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.

7 PSO-GA Approach

PSO-GA is a hybrid of PSO and GA. Harish Garg has proposed a PSO-GA[33] 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[33] 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.

7.1 Parts of PSO-GA

The algorithm for PSO-GA is shown below

1. Set PSO and GA parameters

    - Set current PSO iteration, $PSO_{CurrIt} = 0$ and max iteration PSO_{MaxIt}
    - Set PSO population size PSO_{PopNum}, cognitive and social bias constants c_1 and c_2, maximum and minimum inertial weights w_{max} and w_{min}
    - Set GA parameters, crossover probability GA_{cross}, mutation probability GA_{mut}
    - Set GA parameters: rate of the number of PSO particles affected by GA γ and rate of increasing GA maximum iterations β, maximum and minimum number of individuals to be selected GA_{NumMax} and GA_{NumMin}, maximum and minimum GA population sizes $GA_{MaxPopSize}$ and $GA_{MinPopSize}$, maximum and minimum GA iteration numbers GA_{MinItr} and GA_{MaxItr}
    - Set the PSO dependent GA parameters, number of individuals affected by GA GA_{Num}, GA population size $GA_{PopSize}$ and GA maximum iteration GA_{MaxItr} using the equations

$$GA_{Num} = GA_{NumMax} - (\frac{PSO_{CurrIt}}{PSO_{MaxIt}})^{\gamma} \times (GA_{NumMax} - GA_{NumMin}) \tag{3}$$

$$GA_{PopSize} = GA_{MinPopSize} + (\frac{PSO_{CurrIt}}{PSO_{MaxIt}})^{\gamma} \times (GA_{MaxPopSize} - GA_{MinPopSize}) \tag{4}$$

$$GA_{MaxItr} = GA_{MinItr} + (\frac{PSO_{CurrIt}}{PSO_{MaxIt}})^{\beta} \times (GA_{MaxItr} - GA_{MinItr}) \tag{5}$$

PSO Section

2. Generate a random population of particles of PSO_{PopNum} members in D dimensions, each with a corresponding random velocity v

3. Increment PSO_{CurrIt} by 1

4. Evaluate each particle's objective function value $F(PSOx)$

5. Update *gbest* and *pbest* positions and values of each $PSOx_i$ in the population ($i \in 1, 2, 3, ..., PSO_{PopNum}$)

6. Update each particle's velocity and position with the equations,

$$w = w_{max} - (w_{max} - w_{min}) \times (\frac{PSO_{CurrIt}}{PSO_{MaxIt}}) \tag{6}$$

$$v_i = v_i \times w + c_1 \times rand() \times (pbest_i - PSOx_i) + c_2 \times rand() \times (pbest_g - PSOx_i) \tag{7}$$

where $i \in 1, 2, 3, ..., PSO_{PopNum}$ and g is position/individual in the PSO population that is currently designated as global best ($gbest$) individual

$$PSOx_i = PSOx_i + v_i \tag{8}$$

GA Section

7. Set the number of currently selected individuals $GA_{CurrNum} = 0$

8. Increment $GA_{CurrNum}$ by 1

9. Choose a random position/individual $PSOx_s$ from the PSO population.

10. Generate a random population of $GA_{PopSize}$ individuals in the same D dimensions.

11. Set the first individual GAx_1 in the GA population to be a randomly selected individual $PSOx_s$ from the PSO particle population.

12. Set the current GA iteration $GA_{CurrItr} = 0$

13. Increment $GA_(CurrItr)$ by 1

14. Perform elitism

    - set the replacing individual GA_{rep} as the randomly selected PSO particle $PSOx_s$ if $GA_{CurrNum} = 0$
    - otherwise, check each individual in the current GA population, if $F(GAx_i)$ is less fit than $F(PSOx_s)$, then replace GAx_i with $PSOx_s$

$$GAx_i = \begin{cases} PSOx_s & \text{if } F(PSOx_s) < F(GAx_i) \\ GAx_i & \text{otherwise} \end{cases} \quad i \in 1, 2, ..., GA_{PopSize}$$

15. Perform selection, crossover and mutation to generate the next GA population

16. Evaluate the penalizing objective fitness values $F(GAx_i)$ for each individual in the GA population

17. Check if maximum GA iterations is reached
    - If reached, proceed to step 18
    - otherwise, go back to step 13

18. Replace the selected PSO particle $PSOx_s$ with the best individual in the GA population

19. Check if the maximum number of replacements have occurred
    - If reached, proceed to step 20
    - otherwise, go back to step 9

20. Update the PSO dependent GA parameters using equations (3), (4) and (5)

21. Check if the maximum number of PSO iterations have been reached or if the population has converged
    - If reached, end
    - otherwise, go back to step 3

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}. 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, $GA_{PopSize}$ and GA_{MaxItr} which are for the population size in GA and the maximum iterations done for GA respectively by the equations (3), (4) and (5).

Judging from the equations 3, 4 and 5, GA_{Num} will initially be GA_{NumMax} and slowly become GA_{NumMin} as the number of iterations increases. This is because the fraction PSO_{CurrIt}/PSO_{MaxIt} is raised to γ which is a positive whole number as given by Garg[33], hence, the whole term $(PSO_{CurrIt}/PSO_{MaxIt})^{\gamma}$ will initially be very small and eventually will be equal to 1 when $PSO_{CurrIt} = PSO_{MaxIt}$.

This is also the case for both $GA_{PopSize}$ and GA_{MaxItr}. $GA_{PopSize}$ will initially start equal to $GA_{MinPopSize}$ then slowly become $GA_{MaxPopSize}$. GA_{MaxItr} will initially start equal to GA_{MinItr} then slowly become GA_{MaxItr}. Since the factors will be in fractions, there is a need to get the floor values of GA_{Num}, $GA_{PopSize}$ and GA_{MaxItr}. This is because GA_{Num}, $GA_{PopSize}$ and GA_{MaxItr} must be positive integers because they dictate array sizes. However, in the case of Inertial Weight w, which changes according to the equation

Table 1: Vertices and their Characteristics

Vertices (i)	0	1	2	3	4	5	6	7	8
Load (q_i)	0	2	1.5	4.5	3	1.5	4	2.5	3
Service Time (s_i)	0	1	2	1	3	2	2.5	3	0.8
Time Window $[a_i, b_i]$	[0,14]	[1,4]	[4,6]	[1,2]	[4,7]	[3,5.5]	[2,5]	[5,8]	[1.5,4]

Table 2: Distances from Point to Point

$d_{(i,j)}$	0	1	2	3	4	5	6	7	8
0	0	40	60	75	90	200	100	160	80
1	40	0	65	40	100	50	75	110	100
2	60	65	0	75	100	100	75	75	75
3	75	40	75	0	100	50	90	90	150
4	90	100	100	100	0	100	75	75	100
5	200	50	100	50	100	0	70	90	75
6	100	75	75	90	75	70	0	70	100
7	160	110	75	90	75	90	70	0	100
8	80	100	75	150	100	75	100	100	0

$w = w_{max} - (w_{max} - w_{min})(PSO_{CurrIt}/PSO_{MaxIt})$, it is most of the time a fraction. Garg[33] started $w = 0.9$ initially then becoming $w = 0.4$ as the number of iterations increases.

8 Test Case

We first test if the PSO-GA algorithm works. Liu et.al.[34] worked on a Hyrbid PSO wherein the cross-over mechanism of GA was implemented on the PSO population. Each member of the population was crossed with their respective personal best and the global best positions. They tested the algorithm for a small scale example, they used an undirected graph G whose vertices are given in table 1. Vertex 0 is the depot while the rest are the 8 customers. The distances assigned to the edges are given in table 2. How this works is that since it is an undirected/symmetric graph, $(i, j) = (j, i)$, if we want to know the distance from any two nodes say the depot to the first customer location, we check the values of the table at $d_{(0,1)}$ or $d_{(1,0)} = 40$. The distances are measured yards, the time is measured in hours. Travel time is dictated by distance, it is assumed that all vehicles travel at constant speed of 50 yards per hour. Hence, if a vehicle travels from the depot to the first customer location, the distance traveled is 40 yards while the travel time is given as $40\,\cancel{\text{yards}} \times \frac{1\text{ hour}}{50\,\cancel{\text{yards}}} = 0.8$ hours $= 48$ minutes. Three vehicles are considered in this test case, $K = \{1, 2, 3\}$.

We keep track of when a vehicle $l \in K$ arrives at a customer location ($arrival_{i,l}$). When a vehicle arrives too early at a customer location, it will have to become idle and wait there until the customer is ready to be served. 'Early' pertains to when the vehicle arrives at customer i before the span of it's time window ($a_i > arrival_{i,l}$). We penalize the vehicle for

being too early at customer location, by calculating how much time it waits. **Waiting time** (P_E) of a vehicle at any given customer location is given as $P_{E_{i,l}} = \max\{0, a_i - arrival_{i,l}\}$. When a vehicle arrives too late at a customer location ($b_i < arrival_{i,l}$), it will still service the customer but a penalty is given for every hour after the designated time window. We calculate how much time has lapsed before the vehicle services customer i through **Delay Time** $P_{L_{i,l}} = \max\{0, arrival_{i,l} - b_i\}$.
We want to minimize the total amount of distance traveled, the amount of time all vehicles stay idle and the amount of time all vehicles are late. The cost of travel distance is 1 : 1, that is one yard of distance traveled is equivalent to one cost. The amount of time a vehicle is idle and late are converted into distance so that they can be added to the cost. We know that the total amount of time that all vehicles are idle is given by the summation $\sum_{l=1}^{3}\sum_{i=0}^{8} P_{E_{i,l}}$. We also know that the total amount of time that all vehicles are late is given by the summation $\sum_{l=1}^{3}\sum_{i=0}^{8} P_{L_{i,l}}$. P_E and P_L can be converted into distance given the speed of the vehicles. For example, a vehicle waits for 1 hour, then we convert that into distance by $\cancel{1 \text{ hour}} \times \frac{50 \text{ yards}}{\cancel{1 \text{ hour}}} = 50$ yards. Our cost function is therefore given by the equation

$$\text{Total Cost } = \text{Cost}_{\text{distance}} + \text{Cost}_{\text{waiting}} + \text{Cost}_{\text{late}} \tag{9}$$

The known solution to this problem is given by the routes:

- Vehicle 1: $(0 \to 3 \to 1 \to 2 \to 0)$
- Vehicle 2: $(0 \to 8 \to 5 \to 7 \to 0)$
- Vehicle 3: $(0 \to 6 \to 4 \to 0)$

To evaluate these routes, we add the distances traveled by each vehicle:

- Vehicle 1:
  $d_{(0,3)} + d_{(3,1)} + d_{(1,2)} + d_{(2,0)}$
  $75 + 40 + 65 + 60 = 240$
- Vehicle 2:
  $d_{(0,8)} + d_{(8,5)} + d_{(5,7)} + d_{(7,0)}$
  $80 + 75 + 90 + 160 = 405$
- Vehicle 3:
  $d_{(0,6)} + d_{(6,4)} + d_{(4,0)}$
  $100 + 75 + 90 = 265$

Thus, the total distance traveled by all vehicles is 910 yards. Next we compute how much waiting time and delay time are experienced by each vehicle in their respective routes

- Vehicle 1:

$$arrival_{0,1} = 0$$

$$P_{E_{0,1}} = \max\{0, (a_0 - arrival_{0,1})\} = \max\{0, 0\} = 0$$

$$P_{L_{0,1}} = \max\{0, (arrival_{0,1} - b_0)\} = \max\{0, -14\} = 0$$

$$arrival_{3,1} = \max\{a_0, arrival_{0,1}\} + d_{(0,3)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_0$$

$$= 0 + \frac{75}{50} + 0 = 1.5 \text{ hours}$$

$$P_{E_{3,1}} = \max\{0, (a_3 - arrival_{3,1})\} = \max\{0, -0.5\} = 0$$

$$P_{L_{3,1}} = \max\{0, (arrival_{3,1} - b_3)\} = \max\{0, -0.5\} = 0$$

$$arrival_{1,1} = \max\{a_3, arrival_{3,1}\} + d_{(3,1)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_3$$

$$= 1.5 + \frac{40}{50} + 1 = 3.3 \text{ hours}$$

$$P_{E_{1,1}} = \max\{0, (a_1 - arrival_{1,1})\} = \max\{0, -2.3\} = 0$$

$$P_{L_{1,1}} = \max\{0, (arrival_{1,1} - b_1)\} = \max\{0, -0.7\} = 0$$

$$arrival_{2,1} = \max\{a_1, arrival_{1,1}\} + d_{(1,2)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_1$$

$$= 3.3 + \frac{65}{50} + 1 = 5.6 \text{ hours}$$

$$P_{E_{2,1}} = \max\{0, (a_2 - arrival_{2,1})\} = \max\{0, -1.6\} = 0$$

$$P_{L_{2,1}} = \max\{0, (arrival_{2,1} - b_2)\} = \max\{0, -0.4\} = 0$$

$$arrival_{0,1} = \max\{a_2, arrival_{2,1}\} + d_{(2,0)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_2$$

$$= 5.6 + \frac{60}{50} + 2 = 7.0 \text{ hours}$$

$$P_{E_{0,1}} = \max\{0, (a_0 - arrival_{0,1})\} = \max\{0, -7.0\} = 0$$

$$P_{L_{0,1}} = \max\{0, (arrival_{0,1} - b_0)\} = \max\{0, -7.0\} = 0$$

- Vehicle 2:

$$
\begin{aligned}
arrival_{0,2} &= 0\\
&\quad P_{E_{0,2}} = \max\{0, (a_0 - arrival_{0,2})\} = \max\{0, 0\} = 0\\
&\quad P_{L_{0,2}} = \max\{0, (arrival_{0,2} - b_0)\} = \max\{0, -14\} = 0\\
arrival_{8,2} &= \max\{a_0, arrival_{0,2}\} + d_{(0,8)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_0\\
&= 0 + \frac{80}{50} + 0 = 1.6 \text{ hours}\\
&\quad P_{E_{8,2}} = \max\{0, (a_8 - arrival_{8,2})\} = \max\{0, -0.1\} = 0\\
&\quad P_{L_{8,2}} = \max\{0, (arrival_{8,2} - b_8)\} = \max\{0, -2.4\} = 0\\
arrival_{5,2} &= \max\{a_8, arrival_{8,2}\} + d_{(8,5)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_8\\
&= 1.6 + \frac{75}{50} + 0.8 = 3.9 \text{ hours}\\
&\quad P_{E_{5,2}} = \max\{0, (a_5 - arrival_{5,2})\} = \max\{0, -0.9\} = 0\\
&\quad P_{L_{5,2}} = \max\{0, (arrival_{5,2} - b_5)\} = \max\{0, -1.1\} = 0\\
arrival_{7,2} &= \max\{a_5, arrival_{5,2}\} + d_{(5,7)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_5\\
&= 3.9 + \frac{90}{50} + 2 = 7.7 \text{ hours}\\
&\quad P_{E_{7,2}} = \max\{0, (a_7 - arrival_{7,2})\} = \max\{0, -2.7\} = 0\\
&\quad P_{L_{7,2}} = \max\{0, (arrival_{7,2} - b_7)\} = \max\{0, -0.3\} = 0\\
arrival_{0,2} &= \max\{a_7, arrival_{7,2}\} + d_{(7,0)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_7\\
&= 7.7 + \frac{160}{50} + 3 = 13.9 \text{ hours}\\
&\quad P_{E_{0,2}} = \max\{0, (a_0 - arrival_{0,2})\} = \max\{0, -13.9\} = 0\\
&\quad P_{L_{0,2}} = \max\{0, (arrival_{0,2} - b_0)\} = \max\{0, -0.1\} = 0
\end{aligned}
$$

- Vehicle 3:

$$
\begin{aligned}
arrival_{0,3} &= 0\\
&\quad P_{E_{0,3}} = \max\{0,(a_0 - arrival_{0,3})\} = \max\{0,0\} = 0\\
&\quad P_{L_{0,3}} = \max\{0,(arrival_{0,3} - b_0)\} = \max\{0,-14\} = 0\\
arrival_{6,3} &= \max\{a_0, arrival_{0,3}\} + d_{(0,6)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_0\\
&= 0 + \frac{100}{50} + 0 = 2 \text{ hours}\\
&\quad P_{E_{6,3}} = \max\{0,(a_6 - arrival_{6,3})\} = \max\{0,0\} = 0\\
&\quad P_{L_{6,3}} = \max\{0,(arrival_{6,3} - b_6)\} = \max\{0,-3\} = 0\\
arrival_{4,3} &= \max\{a_6, arrival_{6,3}\} + d_{(6,4)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_6\\
&= 2 + \frac{75}{50} + 2.5 = 6 \text{ hours}\\
&\quad P_{E_{4,3}} = \max\{0,(a_4 - arrival_{4,3})\} = \max\{0,-2\} = 0\\
&\quad P_{L_{4,3}} = \max\{0,(arrival_{4,3} - b_4)\} = \max\{0,-1\} = 0\\
arrival_{0,3} &= \max\{a_2, arrival_{4,3}\} + d_{(4,0)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_4\\
&= 6 + \frac{90}{50} + 3 = 10.8 \text{ hours}\\
&\quad P_{E_{0,3}} = \max\{0,(a_0 - arrival_{0,3})\} = \max\{0,-10.8\} = 0\\
&\quad P_{L_{0,3}} = \max\{0,(arrival_{0,3} - b_0)\} = \max\{0,-3.2\} = 0
\end{aligned}
$$

The total of the penalties are both 0. Thus, out total cost is 910.
The PSO-GA algorithm was used using the following factors:

- PSO population Size = 40
- PSO Maximum Iterations = 200, 400, 800, 1200
- Cognitive and Social factors $c_1 = 1.5$, $c_2 = 1.5$
- Initial and final inertia weight $w_i = 0.9$ $w_f = 0.4$
- Crossover Rate = 0.85
- Mutation Rate = 0.02
- $\gamma = 10$
- $\beta = 15$
- GA Initial Population Size = 10

- GA Final Population Size = 5
- GA Minimum Iterations = 10
- GA Maximum Iterations = 15

The algorithm and problems were encoded and run using Matlab v.2015 on a computer with the following specifications:

$$\begin{aligned} \text{CPU} &= \text{Intel i5-6200U 2.3 GHz} \\ \text{RAM} &= \text{16 Gb} \\ \text{OS} &= \text{Windows 10 Home 2017} \end{aligned}$$

The results are presented in the following table:

Table 3: Distances from Point to Point

Maximum Iterations	Best Value	Worst Value	Number of Converged Runs (out of 10)	Average Running Time (s)	Std Dev.
200	1030	1440	4	61.942182	201.964312
400	960	6285	9	123.734422	1722.612478
800	960	1265	6	326.079594	109.635761
1200	960	1450	9	421.422540	1450.494408

As we can see, the algorithm did not find the optimal solution however, it got the closest possible configuration.

- Vehicle 1: $(0 \to 3 \to 1 \to 2 \to 0)$
- Vehicle 2: $(0 \to 8 \to 5 \to 4 \to 0)$
- Vehicle 3: $(0 \to 6 \to 7 \to 0)$

which has a cost of 960. In analysis of the algorithm, the current problem of not finding the best solution is probably because the random initial populations generated are not that good or there are not enough members for searching. Taking a look at the results, the best solution was found in some of the runs but the population failed to converge. We try to look at bigger population sizes.

9 Problem Formulation

We start with defining some variables. Then we show how each particle or chromosome is decoded into tours(set of routes). From there we define the objective functions and the constraints that will be used in modeling the problem.

The objective in Waste Collection Vehicle Routing Problem with Time Windows (WCVRPTW) is find a set of routes for vehicles, minimizing travel time and distance, such that all consumers are visited exactly once within the time window while satisfying the constraints imposed upon by vehicle capacities.

We first set up some definitions and assumptions essential to creating our model. We represent our network of customers (also called "bins" or "collection sites"), depot(s), and disposal site(s) as a graph $G = (V, E)$ of V vertices and E edges. The set of nodes V represent and encapsulate the set of depot(s) (V^d), consumers (V^c), and disposal site(s) (V^p), that is $V = \{V^d \cup V^c \cup V^p\}$. The size of V (denoted as $|V|$) is therefore computed as the sum of the number of depot(s) ($|V^d| = n_d$), consumers ($|V^c| = n_c$), and disposal site(s) ($|V^p| = n_p$), that is $|V| = |V^d| + |V^c| + |V^p| = n_d + n_c + n_p$. In our case, we consider only one central depot, represented as $V^d = \{0\}$, n_c consumers $V_c = \{1, ..., n_c\}$ and n_p disposal sites $V^p = \{n_c + 1, ..., n_c + n_p\}$, respectively. Each node $\in \{1, 2, ..., n_c\}$ represents each of the consumers taken into consideration. Hence, $|V| = n_c + n_p + 1$. The set of edges $E = \{(i, j)|i, j \in V, i \neq j\}$, that is each $(i, j) \in E$ connects the pair of nodes i and j in V. We can visualize graph G as in figures 16 and 17.

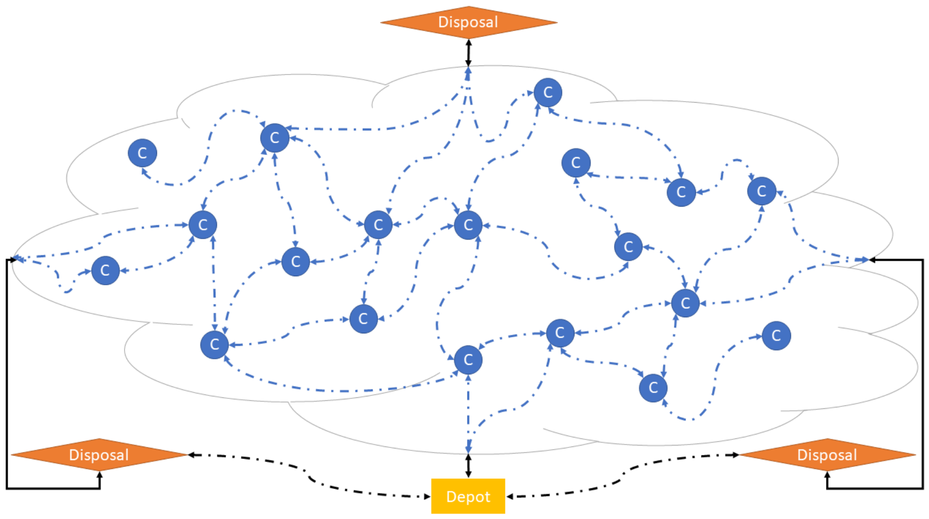


Figure 16: Graph Representative Visualization

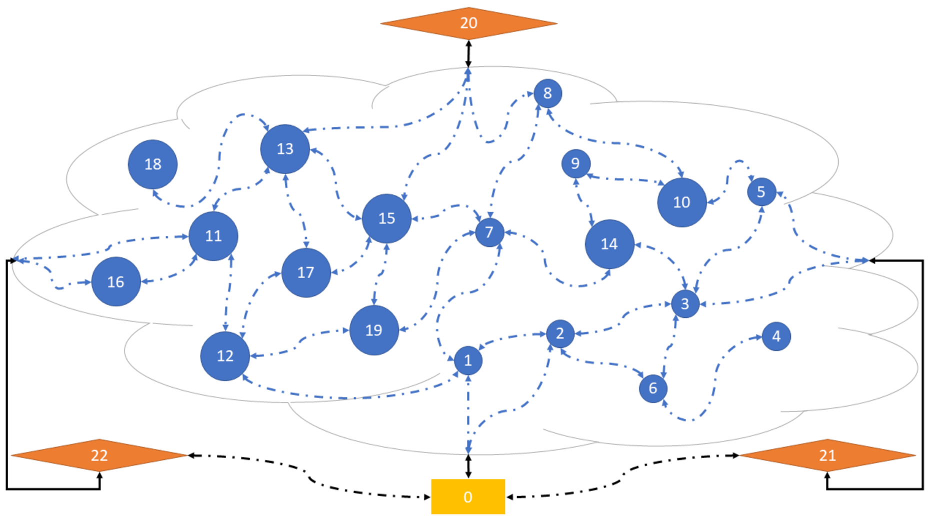


Figure 17: Graph Numerical Visualization

Let $K = \{1, ..., k\}$ be the set of waste collection vehicles and let $time_{(i,j)}$ and $dist_{(i,j)}$ be the travel time and distance associated with edge $(i, j) \in E$, respectively. This means, that if for any vehicle $l \in K$ that traverses edge (i, j), it travels from nodes i to j and covers a distance of $dist_{(i,j)}$ space units in $time_{(i,j)}$ time units. Each node $i \in V$ is associated with service time $serve_i$ and time window $[a_i, b_i]$. We define q_i as the amount of waste collected at a consumer $i \in V^c$. It is assumed that all vehicles can carry at most Q_{cap} amount of waste. As vehicle $l \in K$ traverses it's route, it cumulatively collects the amount of waste for every consumer $i \in V^c$ it services. When the cumulative amount reaches some $90 - 100\%$ of Q_{cap}, the vehicle then travels to a disposal site $i \in V^p$ to dispose of its load and replenish its full carrying capacity. Service time $serv_i$ is the amount of time needed to collect the waste at consumer $i \in V^c$ or the amount of time needed to unload the vehicle at disposal site $i \in V^p$. Note that no waste is collected at either the depot $\{0\}$ and at disposal sites $i \in V^p$ and that the service time at the depot is not necessary, hence $serv_0 = 0$. If any consumer $i \in V^c$ has a greater amount of waste than the maximum carrying capacity of the vehicle, that is $q_i > Q_{cap}$ then we create more instances of consumer i and divide the amount of waste accordingly.

We now identify our assumptions for the model. Some may be repetitions of the above definitions.

1. Each vehicle must start and end at the depot. Also, no garbage is collected at the depot.

2. All our vehicles $l \in K$ are homogeneous, that is, they all have the same model and carrying capacity Q_{cap}.

3. Graph G is connected. That is, all nodes $i \in V$ are reachable from any node $j \in V$. Mathematically stated as "there exists a simple path (made up of edges in E) that connects any pair of nodes $i, j \in V$". It follows that all vehicles $l \in K$ can travel on land to reach any location $u \in V$ using a sequence of nodes $(i, j) \in E$. We do not include locations that cannot be reached or traveled to using our vehicles.

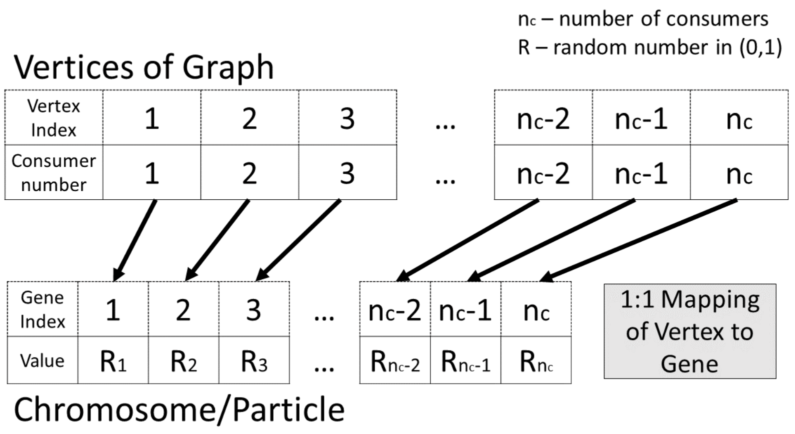


Figure 18: Vertex to Chromosome/Particle Gene

4. Graph G is complete. That is, there exists a path, direct or indirect from any $i \in V$ to all other nodes $j \in V \setminus i$. Also that the path is the shortest possible path from i to j. We can satisfy this using the Floyd-Warshall algorithm or applying the Dijkstra's Shortest Path algorithm for all nodes $i \in V$. We consider this because not all locations in a map can have a direct roadway that connects them, but we can generate an indirect path using known roadways.

5. We consider a divide-and-conquer method wherein we only consider creating a route for each vehicle per day. We proceed with generating routes for each day of the seven days of the week.

6. All values for distance and time for any edge $(i, j) \in E$ is known and that graph G is symmetric.

We now proceed to model the problem. We discuss how we decode or 'make-sense' of the orderable list of consumers given by/and utilized in the PSO-GA algorithm. From the previous section, we defined that each particle or chromosome in the population is composed of some n_c dimensions representing bins or garbage collection sites around the city. Each dimension $dim_f, f \in [1, n]$ is assigned a real number in $(0, 1)$. These real numbers will be used to sort the n_c collection sites and place them into a list similar to the implementation of Masrom[14]. The visualization of these steps are shown on figures 18 and 19. An example of mapping the sorting is shown on figure 20.

We define our decision variable $edgeT_{(i,j),l} \in \{0, 1\}$ is 1 if and only if vehicle $l \in K$ utilizes edge $(i, j) \in E$, otherwise, it is 0. $accumW_{i,l}$ represents the accumulative waste collected at node $i \in V$ for vehicle $l \in K$. $begS_{i,l}$ represents the start time of service at node $i \in V$ for vehicle $l \in K$. $arrival_{i,l}$ represents the time of arrival of any vehicle $l \in K$ at any node $i \in V^c$.

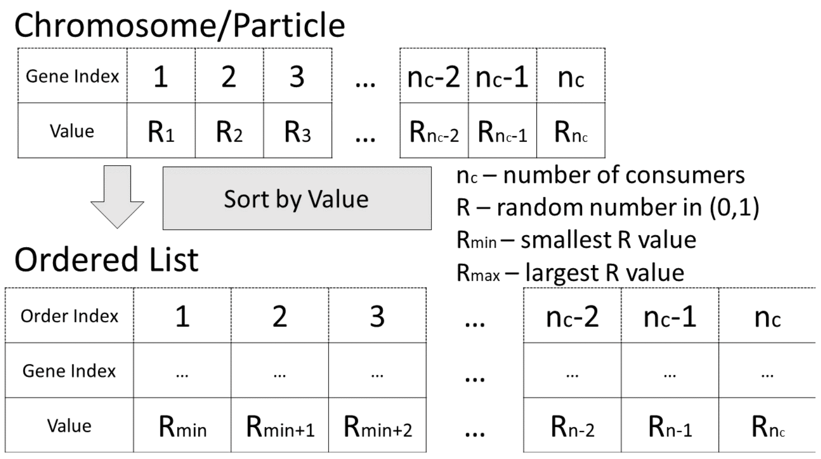

Chromosome/Particle
Gene Index | 1 | 2 | 3 | … | nc-2 | nc-1 | nc
Value | R1 | R2 | R3 | … | Rnc-2 | Rnc-1 | Rnc
Sort by Value
nc – number of consumers
R – random number in (0,1)
Rmin – smallest R value
Rmax – largest R value
Ordered List
Order Index | 1 | 2 | 3 | … | nc-2 | nc-1 | nc
Gene Index | … | … | … | … | … | … | …
Value | Rmin | Rmin+1 | Rmin+2 | … | Rn-2 | Rn-1 | Rnc

Figure 19: Chromosome/Particle to Ordered List

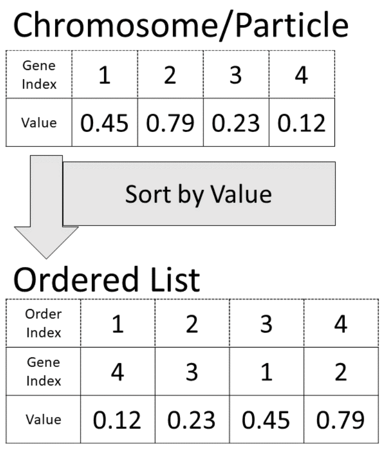

Chromosome/Particle
Gene Index | 1 | 2 | 3 | 4
Value | 0.45 | 0.79 | 0.23 | 0.12
Sort by Value
Ordered List
Order Index | 1 | 2 | 3 | 4
Gene Index | 4 | 3 | 1 | 2
Value | 0.12 | 0.23 | 0.45 | 0.79

Figure 20: Chromosome/Particle to Ordered List Example

References

[1] A. E. Plaza, “Ditch nimby to fix philippines municipal solid waste problem,” *Asian Development Blog*.

[2] D. See, “City to purchase 4 more trucks,” *Sun Star Baguio*.

[3] D. See, “Approval of citys solid waste plan in order,” *Sun Star Baguio*.

[4] F. Olowan, E. Bilog, L. Y. Jr., E. Avila, J. Alangsab, E. Datuin, P. Fianza, L. Farinas, A. Allad-iw, B. Bomogao, and M. Lawana, “Baguio city council okays plastic free ordinance,” *Sun Star Baguio*.

[5] G. B. Dantzig and J. H. Ramser, “The truck dispatching problem,” *Manage. Sci.*, vol. 6, pp. 80–91, Oct. 1959.

[6] S. Sahoo, S. Kim, B.-I. Kim, B. Kraas, and A. Popov Jr., “Routing optimization for waste management,” *Interfaces*, vol. 35, pp. 24–36, Jan. 2005.

[7] A. W. J. Kolen, A. H. G. R. Kan, and H. W. J. M. Trienekens, “Vehicle routing with time windows,” *Operations Research*, vol. 35, pp. 266–273, 1987.

[8] B. Ombuki, B. Ross, and F. Hanshar, “Multi-objective genetic algorithms for vehicle routing problem with time windows,” vol. 24, pp. 17–30, 02 2006.

[9] L. H. Son, “Optimizing municipal solid waste collection using chaotic particle swarm optimization in gis based environments: A case study at danang city, vietnam,” *Expert Systems with Applications*, vol. 41, no. 18, pp. 8062 – 8074, 2014.

[10] M. Akhtar, M. A. Hannan, and H. Basri, “Particle swarm optimization modeling for solid waste collection problem with constraints,” in *2015 IEEE 3rd International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, pp. 1–4, Nov 2015.

[11] K. Buhrkal, A. Larsen, and S. Ropke, “The waste collection vehicle routing problem with time windows in a city logistics context,” *Procedia - Social and Behavioral Sciences*, vol. 39, pp. 241 – 254, 2012. Seventh International Conference on City Logistics which was held on June 7- 9,2011, Mallorca, Spain.

[12] P. Kaur and P. Kaur, “A hybrid pso-ga approach to solve vehicle routing problem,” *International Journal of Engineering Develpment and Research (IJEDR)*, vol. 3, August 2015.

[13] X. Liu, W. Jiang, and J. Xie, “Vehicle routing problem with time windows: A hybrid particle swarm optimization approach,” in *2009 Fifth International Conference on Natural Computation*, vol. 4, pp. 502–506, Aug 2009.

[14] *Hybrid Particle Swarm Optimization for vehicle routing problem with Time Windows*.

[15] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," vol. 35, pp. 254–266, 04 1987.

[16] T. Nuortio, J. Kytjoki, H. Niska, and O. Brysy, "Improved route planning and scheduling of waste collection and transport," *Expert Systems with Applications*, vol. 30, no. 2, pp. 223 – 232, 2006.

[17] J.-F. Cordeau, G. Laporte, M. W. Savelsbergh, and D. Vigo, "Chapter 6 vehicle routing," in *Transportation* (C. Barnhart and G. Laporte, eds.), vol. 14 of *Handbooks in Operations Research and Management Science*, pp. 367 – 428, Elsevier, 2007.

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

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

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

[21] F. Heppner and G. U.

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

[23] 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.

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

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1989.

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

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

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

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