1 Preliminaries

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

1.1 Definitions

1. 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.

2. **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.

3. 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.

4. 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.

5. **Design Variables** are the input to objective functions. 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.

6. 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 be the only factor that may be taken in consideration, it is the most commonly used factor in differentiating the quality of heuristic approaches.

7. **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.

8. An **evolutionary algorithm** is a generic population-based metaheuristic optimization algorithm. It uses mechanisms inspired by biological evolution such as reproduction, mutation, recombination and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of these solutions. We say 'candidate solutions' because all of these individuals may give an acceptable solution but not all of them 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.

9. A **simulation** is a computational model that imitates real world situations and processes. These usually involved an implementation of mathematical equations that employ stochastic variables for a more 'lifelike' appearance.

10. A **stochastic variable** is a variable whose value is a random number usually taken from a uniform distribution from 0 to 1. That is, every number between 0 and 1 has an equal chance to be selected.

11. **Natural Selection** is the process by which organisms with better attributes adapted to the environment tend to increasingly survive and transmit their genetic characteristics through generations. **'Survival of the fittest'** is a phrase by Charles Darwin that describes the mechanism of natural selection. It is best understood as survival through reproductive multiplicity. That is, the more survivability an individual has, the more it is likely to reproduce, hence it's genes are more likely to be transmitted to the next generation.
    In natural selection, there is a variation on traits that is to say that individuals have differences in certain attributes such as height, length, shape etc. It is important to note that not all individuals reproduce to the full potential because the environment has a certain limit to the number of creatures it can sustain. The passing of characteristics or traits from one generation to another is called **heredity**. The more advantageous traits is more commonly passed on and retained because they help the individual or group to survive.

12. Gregor Mendel is known as the father of modern genetics. He discovered the mechanics of heredity or how traits are being passed down from parents to offspring. During cell division, thread-like structures located inside the nucleus of animal and plant cells called **chromosomes** are replicated. These chromosomes contain the genes which dictate the attributes of the individual. They tell how the body is to be built and how it functions.

13. **Recombination** or **Crossover** is the rearrangement of the genetic material by exchanging the same gene subsegments of two chromosomes (one from each parent) which allows for the creation of a new individual that has characteristics similar to those of the father and of the mother. Note that the exchanging process may occur in multiple areas of the strands.

14. **Mutation** on the other hand is the alteration of genes resulting from an error during replication. This results in unique characteristics that may be new from the gene pool of the previous generation. Mutation may be good or bad for the individual but this phenomenon has a low chance of occurring naturally for every generation.

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

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

17. 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 $A = \{1, 2, 3\}$.

18. An object 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) (a) **roster method** where we define all the elements included in a set by listing or enumerating all of them; and

    (b) (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.

19. **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.

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

21. A set with infinite elements is called an **infinite set**, $F = \ldots, 1, 2, 3 \ldots.$ and $|F| = \infty$.

22. 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.

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

24. 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.

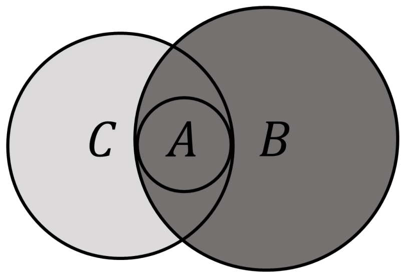

Figure 1: A Venn Diagram Showing the Relationship of A, B and C

25. 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.

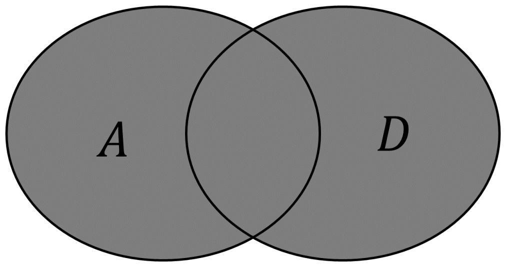

Figure 2: A Venn Diagram Showing $A \cup D$

26. 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.

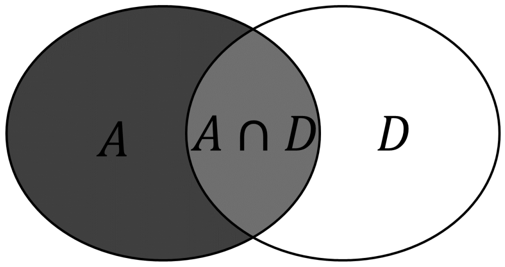

Figure 3: A Venn Diagram Showing $A \cap D$

27. 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.

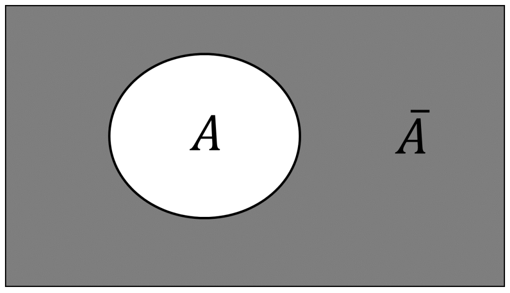

Figure 4: A Venn Diagram Showing $\bar{A}$

28. 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.

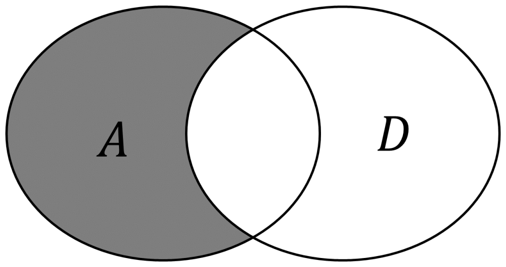

Figure 5: A Venn Diagram Showing $A - D$

29. In mathematics, numbers are grouped in sets and subsets.

    (a) We first have the smallest subset, the set of **Natural** or **Whole Numbers** ($\mathbb{N}$) which is the set of counting numbers, $\{0, 1, 2, 3, 4, 5\ldots\}$.

    (b) The next subset is the set of **Integers** ($\mathbb{Z}$) which is the set of natural numbers and their negatives $\{\ldots$-4, -3, -2, -1, 0, 1, 2, 3, 4, $\ldots\}$.

    (c) Next are the **Rational numbers** ($\mathbb{Q}$) are the ratios of integers, also called fractions, such as $\frac{1}{2}$, $\frac{-10}{56}$ etc.

    (d) 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.

    (e) Finally, the set of **Real Numbers** ($\mathbb{R}$) which consists of both rational and irrational numbers.

    (f) 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 $i = \sqrt{-1}$.

    (g) 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 .

30. A **solution space** the set of all possible values of an optimization problem that satisfy the problem's constraints, potentially including inequalities, equalities, and integer constraints. This is the initial set of candidate solutions to the problem, before the set of candidates has been narrowed down.

31. **candidate solutions** are potential solutions to problems.

32. 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 positions in a sequence, and order of which the elements are enumerated matters, that is if we have two sequences $(1, 2, 3)$ and $(3, 2, 1, 1)$, $(1, 2, 3) \neq (3, 2, 1, 1)$. A sequence is usually denoted by parentheses ('(' and ')'), for example, the famous Fibonacci sequence is

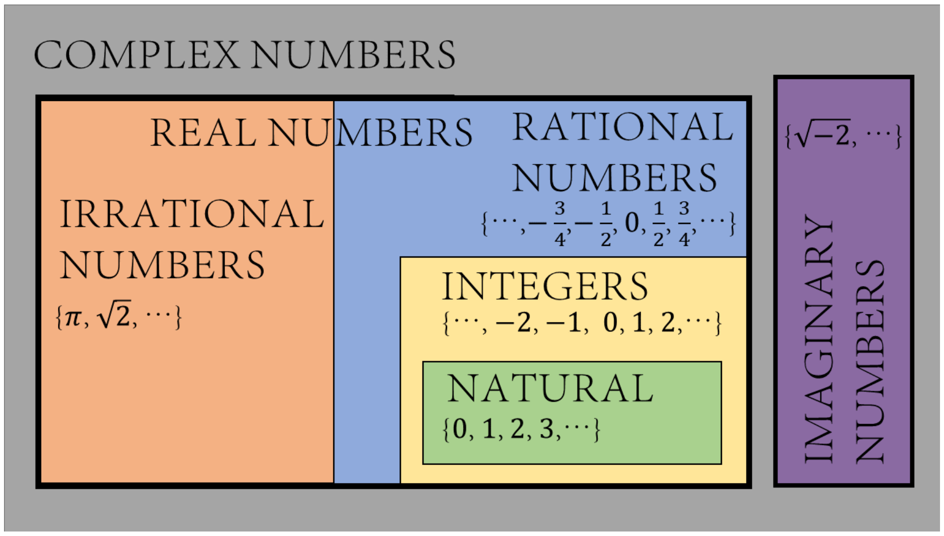
COMPLEX NUMBERS
REAL NUMBERS
RATIONAL NUMBERS
$\{\cdots, -\frac{3}{4}, -\frac{1}{2}, 0, \frac{1}{2}, \frac{3}{4}, \cdots\}$
IRRATIONAL NUMBERS
$\{\pi, \sqrt{2}, \cdots\}$
INTEGERS
$\{\cdots, -2, -1, 0, 1, 2, \cdots\}$
NATURAL
$\{0, 1, 2, 3, \cdots\}$
$\{\sqrt{-2}, \cdots\}$
IMAGINARY NUMBERS

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

given as $(0, 1, 1, 2, 3, 5, 8, \dots)$. Mathematical objects, functions or relations are usually described as sequences.

33. The elements of a sequence are called **terms**.

34. The number of elements of a sequence is called the **length** of that sequence.

35. A sequence may be **finite** in length (ex. $(1, 2, 3, 4, 5)$) or **infinite** (ex. $(1, 2, 3, \dots)$) as in sets.

36. Similar to sets, we can define inclusion to a sequence by:

    (a) The **roster** method, generating all its elements, we must be sure that the sequence is finite.

    (b) 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', $(0, 1, 0, 1, 0, 1, \dots)$.

    (c) 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, $(3, 5, 7, 9, \dots)$.

37. 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".

38. 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:

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

39. A **point** is a location. It has neither width nor length, even though it is visually represented as a dot for reference.

40. Locations are usually made up of a sequence of numbers called **coordinates**.

41. 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 $\overleftrightarrow{AB}$.

42. 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.

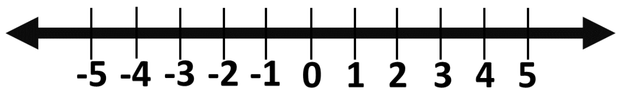

Figure 7: A Real Number Line

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

44. We say that a set of points are **collinear** if there is a line that passes through all the points.

45. A **plane** is a two-dimensional surface. Ruled and spanned by two independent perpendicular lines. A plane is defined by three non-collinear points.

46. 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.

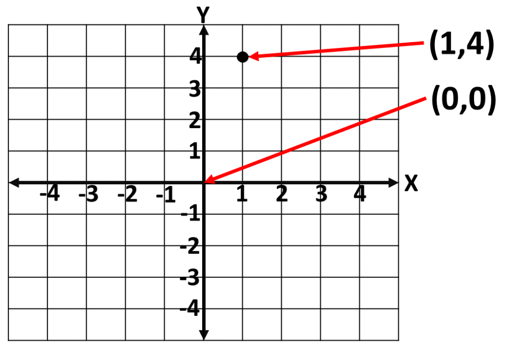

Figure 8: A Coordinate Plane

47. 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 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 $< x_1, x_2, x_3, \dots x_n >$ where each component x_i is the absolute numerical distance between two points in dimension $i \in (1, 2, \dots, n)$. 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. $|a| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$ where each x_i is the component of the vector in dimension $i \in (1, 2, \dots, n)$.

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

49. 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

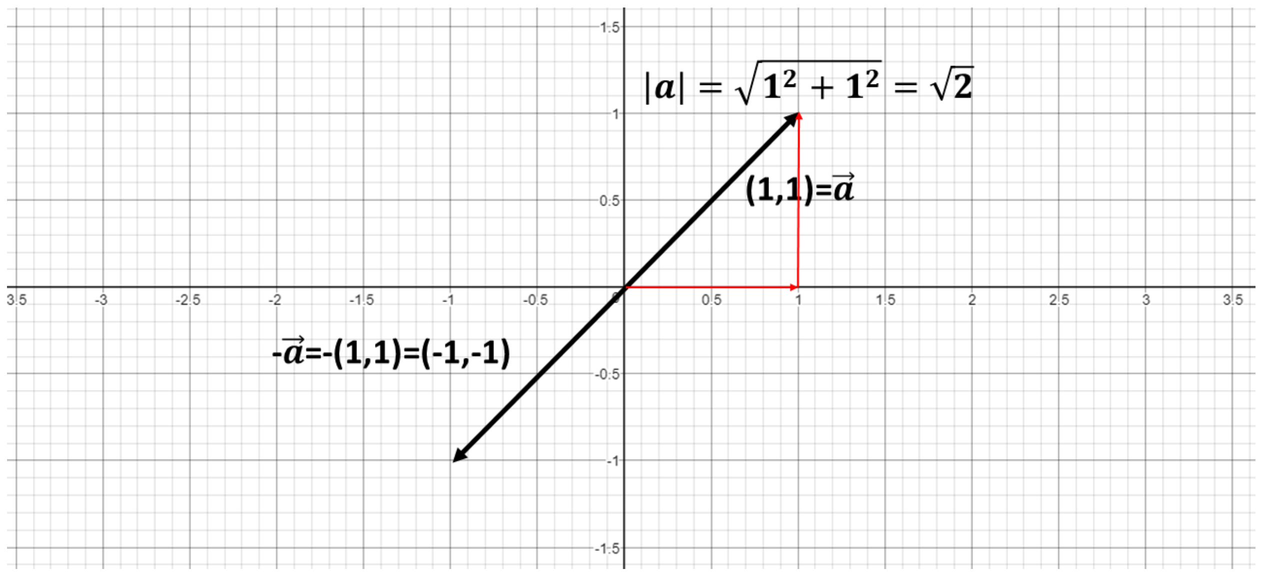

Figure 9: Vectors in a Coordinate Plane

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.

50. **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.

51. 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

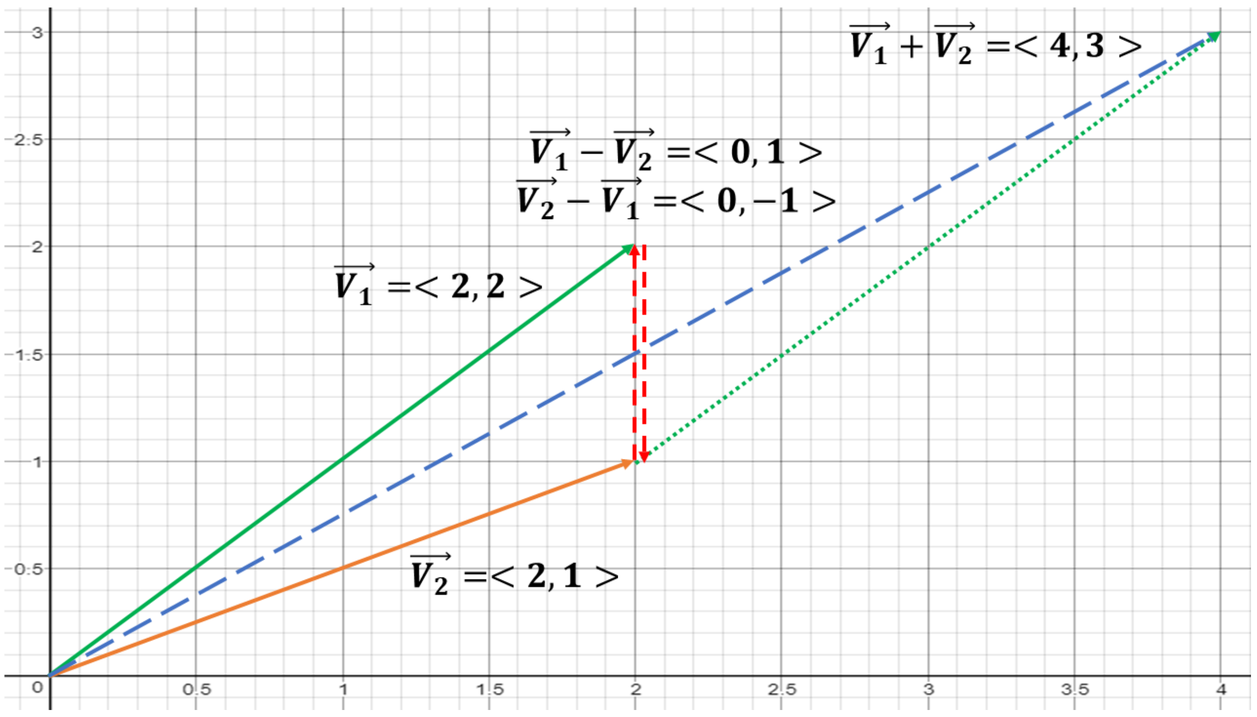

Figure 10: Adding and Subtracting Vectors (Coordinate Plane)

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

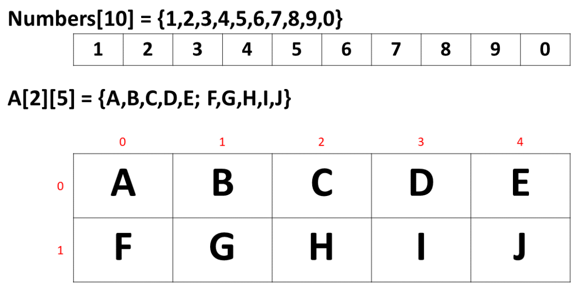
Numbers[10] = {1,2,3,4,5,6,7,8,9,0}
1 2 3 4 5 6 7 8 9 0
A[2][5] = {A,B,C,D,E; F,G,H,I,J}
0 1 2 3 4
0 A B C D E
1 F G H I J

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.

52. A **matrix** is an array of numbers.

The **dimensions** of a matrix is the number of rows and columns of the matrix in that order. A 'two by three' matrix is an array with two rows and three columns. A 'three by two' matrix is an array with three rows and two columns. To show this, we let $M1$ be a 2×3 matrix and $M2$ be a 3×2 matrix.

$$M1 = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}, \ M2 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

We access the elements of a matrix the same way as we do for arrays. An example is $M1[1][1] = a_{11}$

A matrix whose row and column have the same dimension is called a **square matrix**.
The operations that can be done for matrices are as follows:

[**Matrix Addition**] Adding two matrices means that we add their corresponding elements. We can only add matrices with the same dimensions. Let two matrices $M3$ and $M4$ be matrices of the same size $n \times m$, then matrix addition $M3 + M4$ is done as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1m}+b_{1m} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2m}+b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \dots & a_{nm}+b_{nm} \end{bmatrix}$$

[**Multiply by a Constant**] Multiplying a constant number c to a matrix M is done by multiplying the constant to every element of the matrix.

$$c \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix} = \begin{bmatrix} c \cdot a_{11} & c \cdot a_{12} & c \cdot a_{13} & \dots & c \cdot a_{1m} \\ c \cdot a_{21} & c \cdot a_{22} & c \cdot a_{23} & \dots & c \cdot a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c \cdot a_{n1} & c \cdot a_{n2} & c \cdot a_{n3} & \dots & c \cdot a_{nm} \end{bmatrix}$$

[**Negative of a Matrix**] The negative of a matrix is just the matrix multiplied to the constant $c = -1$ hence, all elements are multiplied to -1.

$$-\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{bmatrix} = \begin{bmatrix} -1 \cdot a_{11} & -1 \cdot a_{12} & -1 \cdot a_{13} & \dots & -1 \cdot a_{1m} \\ -1 \cdot a_{21} & -1 \cdot a_{22} & -1 \cdot a_{23} & \dots & -1 \cdot a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 \cdot a_{n1} & -1 \cdot a_{n2} & -1 \cdot a_{n3} & \dots & -1 \cdot a_{nm} \end{bmatrix}$$

[**Matrix Subtraction**] Matrix subtraction is just the addition of two matrices where the addend is negative. Note that here, we can only subtract matrices with the same dimensions. Let two matrices $M3$ and $M4$ be matrices of the same size $n \times m$, then matrix subtraction $M3 - M4$ is done as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} - \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11} - b_{11} & a_{12} - b_{12} & \dots & a_{1m} - b_{1m} \\ a_{21} - b_{21} & a_{22} - b_{22} & \dots & a_{2m} - b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} - b_{n1} & a_{n2} - b_{n2} & \dots & a_{nm} - b_{nm} \end{bmatrix}$$

[**Hadarmard Product**] The Hadamard Product is a **component/element-wise multiplication** where each element is multiplied to the corresponding element of the other matrix. Note that here, we can only multiply matrices with the same dimensions. Let two matrices $M3$ and $M4$ be matrices of the same size $n \times m$, then their hadamard product is given by $M3 \circ M4$ is done as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11} \cdot b_{11} & a_{12} \cdot b_{12} & \dots & a_{1m} \cdot b_{1m} \\ a_{21} \cdot b_{21} & a_{22} \cdot b_{22} & \dots & a_{2m} \cdot b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} \cdot b_{n1} & a_{n2} \cdot b_{n2} & \dots & a_{nm} \cdot b_{nm} \end{bmatrix}$$

[**Matrix Multiplication**] Matrix multiplication is not the Hadamard product. Matrix multiplication involves the sum of products. If A is an $n \times m$ matrix and B is an $m \times p$ matrix, their matrix product AB is an $n \times p$ matrix, in which the m elements across a row of A are multiplied with the m elements down a column of B, the resulting elements are then summed to produce an entry of AB. Let two matrices A and B be matrices of the sizes $n \times m$ and $m \times p$ respectively, then their matrix product is given by $A \times B$ is done as

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mp} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1p} \\ c_{21} & c_{22} & \dots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{np} \end{bmatrix}$$

such that

$$c_{ij} = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + \dots + a_{im} \cdot b_{mj} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}, \forall i = 1, 2, \dots, n \text{ and } j = 1, 2, \dots, p$$

An example is

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = \begin{bmatrix} z_{11} \end{bmatrix}$$

$$z_{11} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

53. The **transpose** of a matrix (denoted as M^T) is a matrix where the rows and columns are swapped. That is

$$M3^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1m} & a_{2m} & \dots & a_{mn} \end{bmatrix}$$

An example is

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \; A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

54. A matrix is said to be **symmetric** if and only if the matrix M is equal to it's transpose M^T. Given by $M = M^T$. An example is

$$O = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix} = O^T = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}, \therefore O \text{ is symmertric}$$

55. 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 $G = (V, E)$.

56. 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).

57. **Edges** are visually seen as lines that connect vertices, they show that those vertices are related in some way. Edges usually connect two vertices, they represent and show that there exists a relationship between these vertices. If there are no edges that connect a pair of vertices, then it an be said that there is no direct relationship between those edges.

58. 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 connected by the **undirected edge** (u, v) (or (v, u)).

59. We also say that the vertices $u, v \in V$ are **adjacent** because an undirected edge connects them.

60. A graph G is called an **undirected graph** if and only if it is made up of undirected edges.

61. 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').

62. 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.

63. A graph G is called a **directed graph** if and only if it is made up of directed edges.

64. 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 $\exists(u, v) \in E \; \forall u, v \in V$).

65. A graph is said to be a **weighted graph** if numbers are assigned to it's edges. These numbers are called **weights** or **costs**.

66. 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.

67. If all vertices of a path are distinct, then the path is said to be **simple**.

68. The **length** of a path is the total number of edges in the path.

69. 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.

70. 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.

71. 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.

72. A graph with no cycles is said to be **acyclic**.

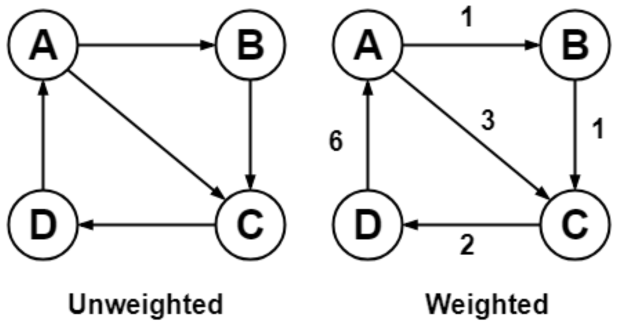

Figure 12: Sample Graph

73. An **adjacency matrix** is a square matrix that shows the relationships of vertices in a graph. Each dimension in the matrix is assigned a vertex. The elements of the matrix is from the set 0, 1. The element $M[u][v] = 1$ if there is an edge that connects vertices u

and v, otherwise, is it 0. The unweighted graph in figure 12 has the adjacency matrix:

$$\begin{array}{c c} & \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

If the graph has weights then we replace the 1's with their respective weights. The adjacency matrix of the weighted graph in figure 12 is:

$$\begin{array}{c c} & \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{bmatrix} 0 & 1 & 3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \\ 6 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Note that for an undirected graph, the adjacency list is symmetric.

74. 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 of 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 an algorithm for solving the single-source shortest path, which means that it solves the shortest path from any node $u \in V$ to any other node $v \in V$. The dijkstra's algorithm uses a priority queue.

A **queue** is a list where the elements are inserted at one end and are removed at the other.

A **priority queue** is a queue wherein each element is associated with a value which dictates whether or not that element is highly likely to be selected/removed from the queue. An element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue.
The dijkstra's algorithm is:

i. Select a source vertex s from the set of vertices V
ii. Create an empty priority queue Q
iii. For each vertex v in the Graph, do the following
   - Set the distance from the source s to vertex v as infinity (∞)
   - Set the optimal path from the source s to node v as empty
   - Add vertex v to the priority queue Q
iv. Set the distance of vertex s from itself as 0

v. While Q is not empty, do the following:
    - Select vertex u from the priority queue Q with the minimum distance
    - Remove u from the queue
    - For each vertex w, still in the queue, adjacent to u, do the following:
        - Compute the path from the source vertex s to the node w that passes through u before it reaches w
        - If the newly computed path is shorter than the current one,
          Update the distance from the source vertex s to vertex w
          Add vertex u to the path of w

The flowchart of the algorithm is seen on figure 13.

(b) **Floyd-Warshall Algorithm** which solves the shortest path for any two node u and v in V. The floyd-warshall algorithm starts off with the adjacency matrix of the graph G. All non-existent edges have the value of infinity ∞. The algorithm takes advantage of the transitivity in order to replace the infinite values. Transitivity is the relation wherein if a property holds between the first and the second and also holds between the second and the third, then it follows that this property also hold between the first and the third. It can be simplified as "if one can go from a to b and from b to c then one can go from a to c by passing through b."
The Floyd-Warshall algorithm is as follows:

i. Let *dist* be a matrix of size $|V| \times |V|$ whose values are ∞
ii. For each edge, $(u, v) \in E$, set $dist_{u,v}$ as the weight of the edge (u, v).
iii. For each vertex $v \in V$, set the distance to itself as 0. $dist_{u,v} = 0$
iv. For each vertex $w \in V$, do the following:
    - For each pair of vertices $u, v \in V$ do the following:
        - Check if the distance from u to v is greater than the distance from u to w and w to v. That is, check if $dist_{u,v} > dist_{u,w} + dist_{w,v}$
        - If that is true, set $dist_{u,v} = dist_{u,w} + dist_{w,v}$

The flowchart of the floy-warshall algorithm is seen on figure 14. An example is shown on figure 15.

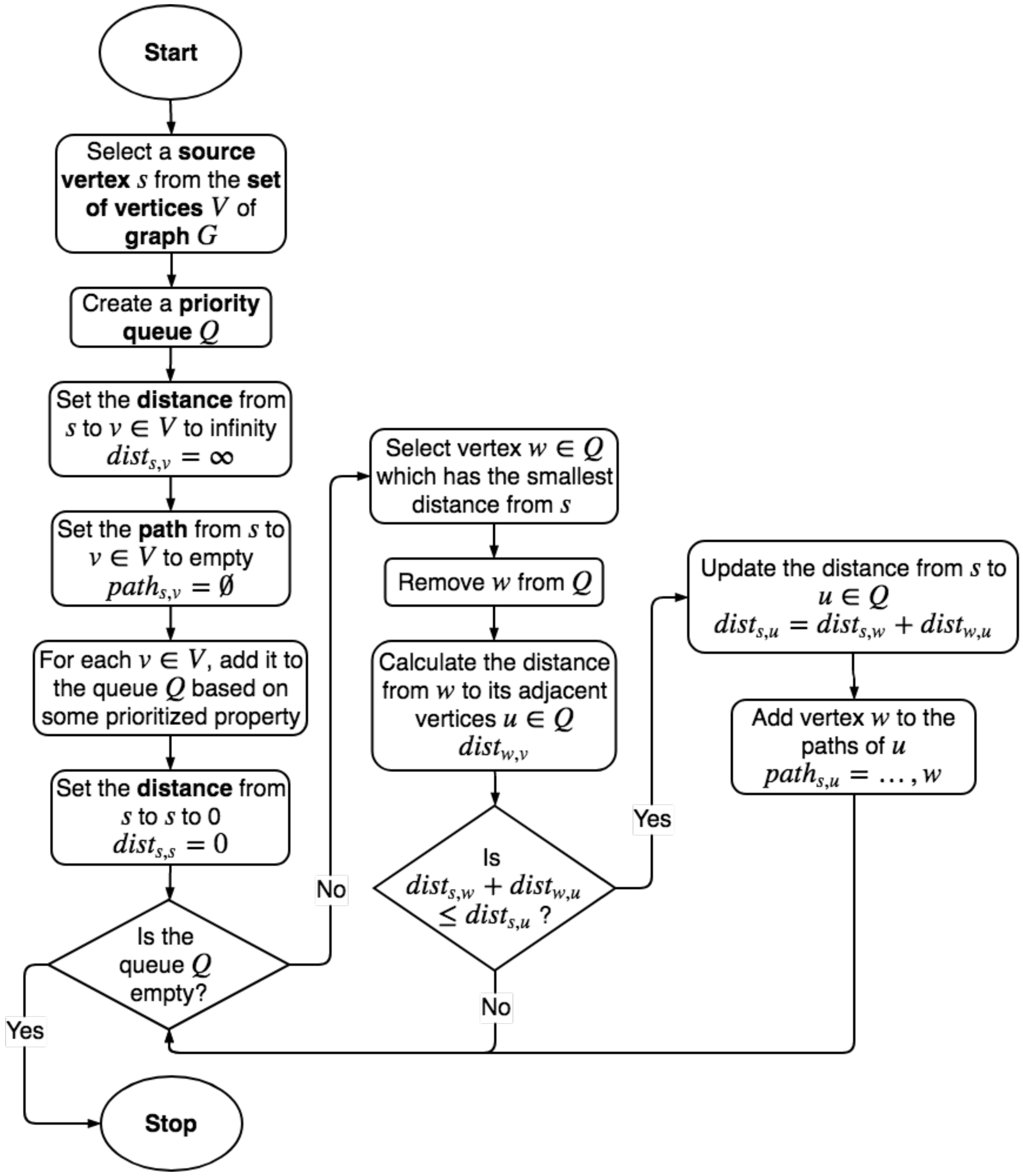
Start
Select a **source vertex** s from the **set of vertices** V of **graph** G
Create a **priority queue** Q
Set the **distance** from s to $v \in V$ to infinity $dist_{s,v} = \infty$
Set the **path** from s to $v \in V$ to empty $path_{s,v} = \emptyset$
For each $v \in V$, add it to the queue Q based on some prioritized property
Set the **distance** from s to s to 0 $dist_{s,s} = 0$
Is the queue Q empty?
Yes
No
Stop
Select vertex $w \in Q$ which has the smallest distance from s
Remove w from Q
Calculate the distance from w to its adjacent vertices $u \in Q$ $dist_{w,v}$
Is $dist_{s,w} + dist_{w,u} \leq dist_{s,u}$?
Yes
No
Update the distance from s to $u \in Q$ $dist_{s,u} = dist_{s,w} + dist_{w,u}$
Add vertex w to the paths of u $path_{s,u} = \ldots, w$

Figure 13: Flowchart of the Dijkstra Algorithm

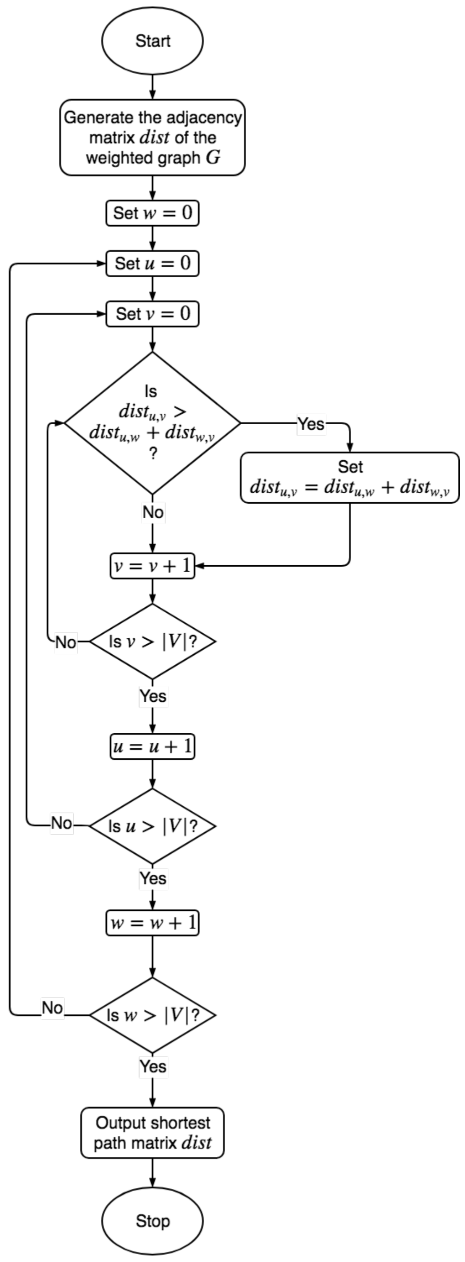
Start
Generate the adjacency matrix $dist$ of the weighted graph G
Set $w = 0$
Set $u = 0$
Set $v = 0$
Is $dist_{u,v} >$ $dist_{u,w} + dist_{w,v}$?
Yes
Set $dist_{u,v} = dist_{u,w} + dist_{w,v}$
No
$v = v + 1$
Is $v > |V|$?
No
Yes
$u = u + 1$
Is $u > |V|$?
No
Yes
$w = w + 1$
Is $w > |V|$?
No
Yes
Output shortest path matrix $dist$
Stop

Figure 14: Flowchart of Floyd-Warshall Algorithm

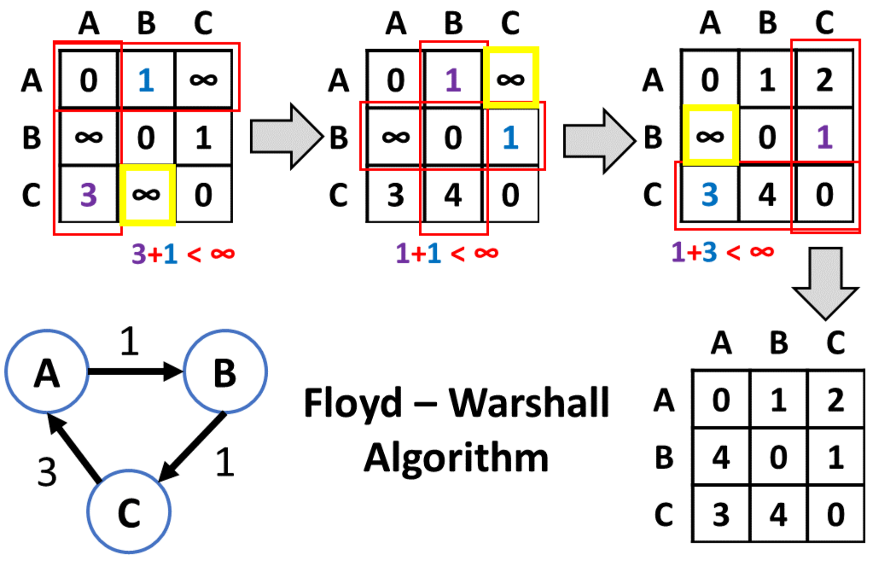

Figure 15: Floyd-Warshall Algorithm Example

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.

75. 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.

76. 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 $V = 0, 1, 2, \ldots, n$ 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.

77. **Waste** is defined as materials that are unwanted or unusable. This refers to matter which are discarded after primary use or is deemed defective, or worthless.

78. There are various types of waste but our focus is one **municipal solid waste** or what we call household trash. This is the type of waste that is produced daily at households and communities. The term 'municipal' comes from the fact that it is the duty of the municipality to collect and manage these kinds of waste. A list of what is considered as municipal solid waste is as follows:

    - **Biodegradable waste** produced from food, cooking, paper etc.
    - **Recyclable materials** such as glass, bottles, jars, clothes, fabrics, rubber etc.
    - **Inert waste** such as dirt, rocks, debris
    - **Electrical and electronic waste** such as appliances, light bulbs, mobile phones, television sets
    - **Composite waste** such as toys, tetra packs, clothing
    - **Hazardous waste** such as paints, batteries, aerosol sprays, and fertilizers
    - **Toxic waste** such as pesticides, herbicides and fungicides
    - **Biomedical waste** such as expired pharmaceutical drugs, used medical equipment

79. **Waste collection** includes gathering, transportation, and delivery for disposal of solid waste and recyclable materials. Waste collection involves vehicles that collect and transport the waste from communities to facilities that receive, sort and process the waste. Processing the received waste may be in the form of incineration, rapid degradation, segregation, resource recovery, energy recover, etc.

80. **Residual waste** is the type of solid waste that is neither recyclable nor reusable.

81. An **Eco-Waste Recovery Services-Material Recovery Facility** is where final sorting of waste is done. Once sorted, the garbage is then moved to designated areas for recycling, recovery, reuse, re-purposing, composting, etc. This facility reduces the amount of residual waste and also corrects any mis-segregated matter.

82. A **Geographic Information System (GIS)** is a collection of computer software, and data used to view, manage, analyze, and transform geographical information. A GIS provides a framework for gathering and organizing spatial data and related information such as temporal, visual, demographic, economic, etc. Out of the data available, it is able to produce analyses, maps, patterns, predictions, assessments and other forms of usable information. It can create fast and logical decisions, produce and display maps, graphs, charts and perform a vast quantity of calculations. An example is Google Maps which offers satellite imagery, street maps, 360 deg panoramic views of streets (Street View), real-time traffic conditions (Google Traffic), and route planning for traveling by foot, car, bicycle (in beta), or public transportation. These kinds of information was produced through available data and some algorithms which processes the data for generating visuals and graphics, route creation, land mark associations etc. Data that is stored in a database is placed in several layers of maps and graphs that have common properties. These layers can come in the form of roadways, vegetation patterns, layout of buildings and structures, traffic information, physical layout of the environment, temperature and pressure maps, radiation maps, demographics, environmental compositions, sets of images and videos, etc. Informally, a physical map is itself a GIS. Within it, is information which can be read and analyzed to produce observations, inferences, hypotheses, predictions, plans, patterns, etc. Basically, it is anything that can tell you something about a place. It has been used in businesses for needs assessments, sales predictions, discovering patterns of customer interests, discovering trends in purchases and demand.

83. **Deterministic** means that the next procedure/step is known without having any other choice. There is no randomness involved.

84. **Non-deterministic** means that there are multiple available decisions that can be done at a certain circumstance. This helps examine the ability to make decisions based on the statements.

85. **Decision problems** is any yes-or-no question that involves an infinite set of inputs. These inputs are logical objects, be it numbers, graphs, strings, or sets. The input is broken down to its properties and based on those properties, the question is thrown an affirmation or negation. For example, "is 4 an element of $\mathbb{Z}$? Well we can compare all numbers in $\mathbb{Z}$ to 4 and this will of course be true, hence the answer returned is 'yes' 4 is an element of $\mathbb{Z}$.

86. Nondeterminstic Polynomial time **NP** is a set of problems with the same resource-based complexity used to describe certain types of decision problems. NP is the set of all decision problems for which the instances where the answer is "yes" are efficiently verifiable through deterministic computations that can be performed in polynomial time. A problem belongs to the $NP - hard$ or the set of the "hardest" NP problems if there is no known polynomial time algorithm that can provide an optimal solution.

2 Introduction

Municipal solid waste management is one of the key services in urban places around the world. It is without a doubt that urbanization and development comes with large volumes of waste. In densely populated cities waste management becomes more difficult as there are increasingly larger amounts of municipal solid waste that needed to be collected. This is because as the standards of living and disposable incomes increases, people are likely to increase their consumption of goods and services, thereby resulting in a corresponding increase in the amount of waste generated. Couple this with the fact that major economies run on an unyielding cycle of production and consumption, waste management demands continuous monitoring. According to World Bank back in 2012, urban population annually produced about 1.3 billion tonnes of Municipal Solid Waste (MSW) which is expected to grow to about 2.2 billion tonnes in 2025.[1]

Waste management in underdeveloped countries are worse since there are few to no proper facilities or vehicles that are needed to handle waste collection efficiently. Poorly managed waste can result to enormous impacts on the environment, health and economy. If waste is left uncollected, it leads to sewer flooding, pollution and road blockades. In contrast, a city that has an efficient waste management would be able to develop faster as its focus shifts to other community needs such as transportation, health, and education.

Waste can be collected in several ways. House-to-house collection is done by individually visiting each house and collecting the garbage straight from the source. Communities can opt for pooling their garbage to certain fixed locations or community bins in the neighborhood. Another way is through curbside pick-ups where households leave their garbage directly out of their houses to be picked-up at a particular time. Households can also volunteer to personally deliver their garbage directly to disposal sites. The local government can opt for waste collection services with private companies. Each of these different options have different ways of computing cost. Costs in waste collection may involve staff salaries, vehicle purchases, fuel costs, buildings and infrastructure, fuel expenditure, maintenance, land use and purchase, business contracts, environmental and health care expenditures, social acceptability, etc.

Before collection, households can be asked to segregate their garbage into specified categories. These categories might include, wet and dry, biodegradable and non-biodegradable, reusable, recyclable, paper, glass, plastics, aluminum. The quality of the waste segregation can determine how efficient and effective the waste processing can be. Certain waste can be re-segregated by a machine-sorter or by designated personnel such as 'pickers'. The quality of how reusable an object is can be determined by the machine or person inspecting the object. Waste such as bear bottles and plastic-containers can be taken out and deemed reusable. Waste such as broken plastic frames or metals can undergo secondary use through remelting and remolding. Papers and plastics can be recycled and re-purposed. Biodegradable waste can be composted and degraded through anaerobic digestion. The rest can be piled into landfills or disintegrated using incinerators. Landfills, however, come with other sets of problem such as health-care, contamination, pollution, land use etc. Incinerating waste increases greenhouse gases, may rise health care issues and some materials may not even be possible to burn. Toxic waste disposal is also an arduous task since harmful chemicals are involved. One cannot just dump them in a field and hope for the best. The more developed the facilities and methods are, the better the waste management. However, it is ironic that urbanization and development the key to solving waste management problems but also correlates to larger volumes of waste produced.

We now take a look at the situation at Baguio City. Every day, about 35,000 tons of municipal solid waste are produced in the Philippines. In Baguio City alone, the amount of garbage produced daily increased by more than 20% from 2008 to 2013.[2] In 2015, 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] As of 2018, Baguio City has 14 functioning waste collection trucks with a pending purchase of 4 more trucks.[4] 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. 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.[5]

In line with these city policies and activities, we study the current efficiency of the routing and scheduling of waste collection vehicles. Specifically, we aim to obtain a set of routes that give the minimum amount of distance while maximizing the quantity of garbage collected. The city currently has 19 functioning vehicles of different kinds and capacities. The vehicles are partitioned such that there is segregation between residual and biodegradable waste, usually a 2:1 partitioning is done. Recyclable materials are usually dealt with by the barangays (villages) who hire personnel to sort the garbage and take out reusable, recyclable materials. The drivers follow a 5-day schedule, the other two days of the week are given as rest days. In each of the five days, they are to service a set of 2-5 barangays. The drivers are set to work 9 hours each working day. Currently, the drivers are the ones who select their routes; they try to attend to the waste collection areas around each barangay while avoiding heavy traffic. All vehicles start at the Eco-Waste Recovery Services-Material Recovery Facility (ERS-MRF) at Barangay Irisan where the drivers sign in for the day. Garbage is collected until vehicle capacities are reach. The residual waste is then transported to the Garbage Transfer Station at Barangay Dontogan. One the other hand, biodegradable, recyclable and reusable materials are brought back to the ERS-MRF for final sorting and composting. It is important to note that there are no time windows that are currently followed since there are too much variables that can affect collection time such as traffic conditions, weather conditions, amount to be collected, etc. The details of the vehicles are seen on table 1. We

Table 1: Vehicle Data

Vehicle	**Type**	**Capacity**	**Number**
Hyundai	Truck	$12m^3$	8
Hino	Truck	$14m^3$	6
Isuzu	10-Wheeler	$18m^3$	3
Daewoo	Compactor	$14m^3$	2

can see that there are more trucks than compactors, these compactors are used as a quick response team for when trucks cannot reach an area or when the trucks have been filled on location and a substantial amount of waste remains.

The usual way of handling the waste collection is by manual creation of routes and schedules through analyzing observational data. Drivers use personal judgment and intuition along the way in trying to avoid any impedance or inconvenience of service. Another way of solving routing and scheduling problems in general is by modeling the problem into a vehicle routing problem as did Dantzig and Ramser[6] in 1959. In the recent decades, vehicle routing has been the focus of some researchers and application developers in solving transportation problems such as deliveries, collections and general transport. Many products of these researches are in the form of Geographical Information Systems such as WasteRoute[7], ArcGIS and Google Earth. Some products are for commercial use such as Waze and Google Maps. From public transportation to product deliveries, vehicle routing solutions has helped not only businesses but also the common man in finding his/her way to their destination. 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.

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 the problem size increases. Some methods under the exact algorithm are the Branch-and-bound method, Dynamic Programming[8], and Integer Programming. Heuristic algorithms on the other hand are able to obtain good approximate solutions to the problem but they may return suboptimal ones. Some methods under the heuristic algorithms include Genetic Algorithms[9], Particle Swarm Optimizations[10, 11], Neighborhood Search Heuristic[12] and some hybrid algorithms[13, 14, 15]. In this study, the problem will be tackled using a hybrid metaheuristic algorithm, Particle Swarm Optimization (PSO) coupled with Genetic Algorithm (GA).

In 2015, Harish Garg proposed a hyrbid PSO-GA algorithm for solving constrained optimization problems. A constrained optimization problem is a problem that is bounded by some limiting factors. Most real-world optimization problems have constraints of different types which modify the shape of the search space. These constraints may come in the form of limited resources, laws and policies imposed upon by a state or country, and the laws of the physical world. During the past years, optimization algorithms have been employed to solve such problems. Constraints can be in the form of both equalities and inequalities, they can be discrete and continuous, linear or non-linear, they can also be related to other constraints. Due to these properties, constrained optimization problems are more difficult to solve compared to unconstrained ones.

According to Garg[16], Constrained optimization problems are defined as:

$$\text{Minimize } F(x)$$

that is subjected to p equality constraints,

$$h_k(x) = 0; \qquad k = 1, 2, \ldots, p$$

and q inequality constraints,

$$g_j(x) \leq 0; \qquad j = 1, 2, \ldots, q$$

where each set of decision variables x is in D dimensions such that $x = [x_1, x_2, x_3, \ldots, x_D]^T$. Each element of x are bounded as

$$l_i \leq x_i \leq u_i; \qquad i = 1, 2, \ldots, D$$

where l_i and u_i are the minimum and maximum permissible values of each element x_i.

In addition, Deb[17] considered that the equality constraints may be converted into inequality constraints since most real world objects are not perfectly accurate in measurement as implied by the uncertainty principle.

Note: 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.

Therefore, equality constraints must be formulated such that the function values of the p equality constraints

$$h_k(x) = 0; \qquad k = 1, 2, \ldots, p$$

must be bounded by some allowable precision δ, that is,

$$|h_k(x)| - \delta \leq 0; \qquad k = 1, 2, \ldots, p$$

Notice now that if we set δ to some precision, say 1×10^{-6}, then it can be said that it is approximately equal to 0. If we increase the precision, then it will be nearer to 0 itself, hence, there would not be a very big difference and therefore it may as well be equal to 0. Since the equality constraints have been converted, we now have the new definition,

$$\text{Minimize } F(x) \tag{1}$$

Subjected to $M = p + q$ inequality constraints,

$$g_j(x) \le 0; \qquad\qquad j = 1, 2, \dots, M$$

where each set of decision variables x is in D dimensions such that $x = [x_1, x_2, x_3, \dots, x_D]^T$. Each element of x are bounded as

$$l_i \le x_i \le u_i; \qquad\qquad i = 1, 2, \dots, D$$

where l_i and u_i are the minimum and maximum permissible values of each element x_i.

We go on to discuss what feasible and infeasible solutions are. Feasible solutions are solutions that do no violate any constraint while infeasible solutions do. There are many approaches in considering feasible and infeasible solutions when implementing optimization algorithms. Some of these methods include rejection of infeasible individuals, maintaining a feasible population, repairing of infeasible individuals, separation of individuals and constraints, replacement of individuals by their repaired versions and use of decoders.[18]

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

He and Wang[19] utilized penalty functions for their Co-evolutionary PSO implementation. They used two groups of swarms. The first group is used to explore the search space while the other group is used to tweak the penalty function parameters. Each swarm in the exploration group is paired with an individual in the parameter group. The individuals in the parameter group determine the penalty functions to be used by the corresponding swarms in the exploration groups. Hence, the solutions obtained in the exploration group depend upon the parameter group while the parameter group depend on the exploration group for evaluation and tweaking. The process aimed to explore and exploit different search spaces in finding the solution.

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

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

where $F(x_i)$ is the penalized objective function value for each individual or particle x_i $i \in 1, 2, 3, \dots, N$ in the population, each x_i must be in the S solution space of D dimensions. $f(x)$ is the non-penalized objective function value of x_i, f_w is the worst objective function value among all x_i individuals and each g_j, $j \in 1, 2, 3, \dots, M$ is the cost or value of each violated constraint. *The solution space of the problem contains all the viable solutions to the problem which also satisfies each and every constraint present.*

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

We will be using the PSO-GA proposed by Harish Garg in solving the vehicle routing problems since the VRP is technically a constrained optimization problem. The constraints related to the vehicle routing problem and it's extensions are the vehicle capacities, time windows, distance or time limits, node count, etc.

3 Capacitated and Waste Collection Vehicle Routing Problems

The VRP is typically a combinatorial problem that deals with arranging multiple specified locations along a single or multiple paths that gives the smallest amount of transportation cost 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 in the real world. In a capacitated VRP, each customer has a certain amount of demand assigned that utilizes the maximum capacity of the vehicles servicing them. Vehicles neither collect nor delivery more than their capacities. VRP models may also include minimizing the number of vehicles needed to satisfy the total customer demand.

The VRP was first introduced by Dantzig and Ramser [20] as the "Truck Dispatching Problem" which is a generalization of the "Traveling Salesman Problem". Given a graph $G = (V, E)$, where $V = v_0, v_1, \ldots, v_n$ is the vertex set and $E = \{(v_i, v_j) : i \neq j, v_i, v_j \in V\}$ is the arc set. Vertex v_0 represented a *depot* at which m homogeneous vehicles are parked, while the remaining vertices correspond to *cities* or *customers*. Moreover, each city v_i has a demand d_i and the total demand of any route may not exceed the homogeneous vehicle capacity Q. A matrix $C = (c_{ij})$ is defined on E, where each element corresponds to the cost of transport across edge (v_i, v_j). The cost represents *distances, travel expenses* or *travel time*. The number of vehicles can be a given constant or decision variable. This means that it can be a fixed number or determined during route construction.

The TSP is concerned with the determination of the shortest route (Hamiltonian Cycle) which passes through each of the n given vertices in the graph G once. It is assumed that there exists some connection between all n vertices. The total number of distinct routes through n vertices is $\frac{1}{2}n!$. In generalizing the TSP, consider constructing several routes, specifically m routes for each of the m vehicles. The goal is to obtain these m routes that generates the minimum cost of transport such that each vehicle starts and ends at the depot, each customer is visited exactly once by one vehicle, and satisfying some other constraints.

Municipal waste collection VRP invovles vehicles collecting municipal waste at customer/bin locations and disposing the load at disposal sites. Vehicles start at a depot and travel along it's route while collecting waste, when full, it then moves to a disposal site to unload the waste collected. The vehicle may then either continue along it's path or return to the depot. WCVRP may impose that waste must either be completely collected or partially collected by a vehicle. Complete collection invokes the rule in VRP that customers are only serviced once. On the other hand, partial collection implies that customers can be serviced more than once by one or more vehicles.

The vehicle routing problem with time windows (VRTPW) is an extension of the VRP problem. Here, some customers identify a time interval at which service is needed or expected to start. This adds time constraints 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 loading and/or unloading processes at each location. 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, hence, these problems are modeled using VRPTW.

Waste Collection VRP may include time windows when customers do impose such constraints. However, in Baguio City, waste management services do not follow time windows as barangays (villages) do not impose such constraints.

4 Review of Related Literature

In 1987, Solomon[21] 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. The process is repeated but this time the 'closest' node from the last added node is obtained and attached until the schedule is full. The algorithm proceeds to create the next route if there are still un-routed nodes. The next heuristic introduced is the insertion heuristic which also constructs routes sequentially. The route starts as two depot nodes. Each customer node is then inserted between two consecutive 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. I1 evaluates the route by distance and start of service time; I2 evaluates the route by total distance and total time; I3 evaluates the route by a combination of total distance, total travel time, and total time vehicles are late. When the node is inserted, the nodes succeeding it in the path 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 into clusters and assigns each cluster to a vehicle. The route construction and scheduling is then done for each cluster of nodes associated to a vehicle. 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 node sequencing rather than grouping and assigning customers to vehicles.

In 2000, Son[10] 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 from a model of the road networks and waste collection system of Danag City. The information used by the GIS are a collection of real data obtained through a span of time. From this data, the average amount of waste collected at an area is known and is then simulated to vary based on the average amount. Traffic and other variables taken into consideration are also simulated the same way. 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 create a schedule that maximizes the amount of garbage collected in the simulation.

In 2005, Nuortio et.al.[22] improved the inflexible and inefficient waste collection scheduling and routing in Eastern Finland by creating a GIS model that is made based on the available road network and waste collection data. They employed a hybrid insertion heuristic for generating the initial population. A guided variable neighborhood thresholding meta heuristic was then used for improving the initial routes. This heuristic is based on three principles, (1) guided local search, which performs a search on the search space S with the intent of finding the local optima. The decision of selecting which part of the search space to explore is based on a deciding factor that 'guides' the search. (2) variable neighborhood search which explores a particular local search space while executing the same local searching approach on adjacent neighborhoods (local search spaces) and switches the current local search space being explored with the neighborhood that shows a better or promising solution. (3) Threshold accepting is a method of evaluating the solutions found and judging whether or not the solution is a good enough approximation of the best solution. This is done for when obtaining the best solution becomes inefficient in terms or resources so instead, it is better to settle for a close approximate. The result of their experimentation showed that the schedule produced by the heuristic significantly reduced traveling distance of vehicles.

In 2007, Cordeau et.al.[23] compiled and defined general models of the vehicle routing problem and it's extensions (i.e. capacitated, time windows, etc.). They also collated and cited several 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.[12] set-up a model for waste collection vehicle routing problem with time windows (WCVRPTW) with lunch breaks based on two test cases, namely that of the (1) Waste Management Inc. which is responsible for waste collection in parts of Northern America and (2) 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. Neighborhood search is a technique that tries to find good or near-optimal solutions to a combinatorial optimization problem by repeated transformation of a current solution into different solutions in its 'neighborhood'. The neighborhood of a solution is a set of similar solutions obtained by relatively simple modifications to the original solution (i.e. swapping two nodes in the route). For a large-scale neighborhood search, the neighborhood produced of a solution is relatively numerous in count compared to the regular one. The 'adoptive' part stems from the fact that the algorithm tries to improve the solution by adjusting the neighborhood produced using the current known solutions at each iteration or time-step. They found that the algorithm produced considerable improved routes from those being used by the two companies.

In 2015, Akhtar, Hannan and Basri[11] proposed a method of solving Waste Collection Vehicle Routing Problem by node clustering in order to simplify the problem. They distributed customers into bins and modeled a traveling salesman problem for each cluster/bin. They then applied the Particle Swarm Optimization algorithm to find optimal routes for each TSP. This method is based on the notion of divide-and-conquer however, note that the clustering method used is significantly important since it determines which nodes go to which route. In their approach, they used smart bin technology which sends information about each bin specifically the .

Masrom et.al.[15] developed a hybrid PSO by incorporating the mutation mechanism of GA. Each particle is made up of $n + 2m$ components where n is the number of customers and m is the number of vehicles. The initial population is made through assigning real numbers to each component. The n components associated with customers are distributed to m vehicles using the real numbers. The customers are assigned to the vehicle whose associated real number is closest to the customers' real number. PSO is followed in each iteration and Mutation only occurs when the population's total health is low. A 'healthy' particle is one that changes it's personal best at each iteration. A population with low total health is a population where the majority of particles have not changed their personal best positions therefore we can say that the population might have fallen into a local optima and has stagnated. Mutation maintains that the population keeps moving even if only at a small distance. Both PSO and GA algorithms are discussed later in this document.

Lu et.al [24] also developed a hybrid PSO algorithm however this time, the crossover mechanism of GA was used. Each particle has $n + l$ components where n is the number of customers and l is the number of vehicles. Integers are assigned to each component which allows for the creation of the initial population. The components are arranged using the integers hence the routes are created. A vehicle's route is the sequence of customer components between two vehicle components. Crossover is done using two particles, a subsequence of the same size is selected in each particle and removed from that particle. The subsequence is then placed at the beginning of the other particle's sequence. This is done for both particles, hence two new sequences are created which replaces the old ones. The results showed that the hyrbid PSO outperformed the basic PSO and basic GA algorithms.

5 Problem Formulation

We start by reevaluating what we know about the current system.

- Each driver works for 9 hours each working day
- Each driver/vehicle is assigned a 5-day work schedule.
- Each vehicle is assigned about 2 to 5 Barangays for servicing at each day.

- There a total of 19 waste collection vehicles. Two (the compactors) of which are quick response teams.
- There are four kinds of vehicles used for waste collection. The details are seen on table 1.
- Each vehicle's carrying capacity has two main partitions, $\frac{2}{3}$ is dedicated for residual waste while the remaining $\frac{1}{3}$ is for biodegradable and other waste.
- Each vehicle start and ends at the depot (Irisan ERS/MRF).
- Each vehicles must be empty at the start and before retiring to the depot.
- There are 129 known Barangays (Villages) in the City.
- The residual garbage is brought to the Garbage Transfer Station.
- The rest of the garbage (mostly biodegradable) is brought back to the ERS/MRF.
- No time windows are alloted due to variability of traffic, road availability, weather conditions, and quantity of waste.
- There are no distinctions between service consumers (i.e. businesses, households, government offices, etc.) hence they will all be considered as collection sites.

A map of all 129 barangays are shown on figure 16. The red house marker indicates the Irisan ERS/MRF, the yellow flag indicates the Dontogan Transfer Station and the blue markers are the 129 barangay markers. The objective in Waste Collection Vehicle Routing Problem is to determine a viable route that minimizes distance or total cost with the following constraints:

1. All vehicles start at and return to the depot;
2. All vehicles are homogeneous, they have the same maximum capacity;
3. A waste collection site is visited by only one vehicle;
4. The total amount of waste collected by vehicle must not exceed its maximum;
5. Distances between the depot, collection sites and the disposal site are determined.
6. The demand at each collection site should be less than the maximum capacity of the vehicle

We represent our network of collection sites, ERS-MRF depot, and disposal site (GTS) as an undirected and complete graph $G = (V, E)$ of V vertices and E edges.

The set of vertices V encapsulates the set of waste collection sites (V^c), the single depot (V^d), and the single disposal site (V^p), that is $V = \{V^d \cup V^c \cup V^p\}$. The number of vertices is therefore $|V| = |V^d| + |V^c| + |V^p| = 1 + n + 1 = n + 2$ where n is the number of waste collection sites.

$$V = \{v_i\}, i \in 0, 1, 2, \ldots, n + 1$$

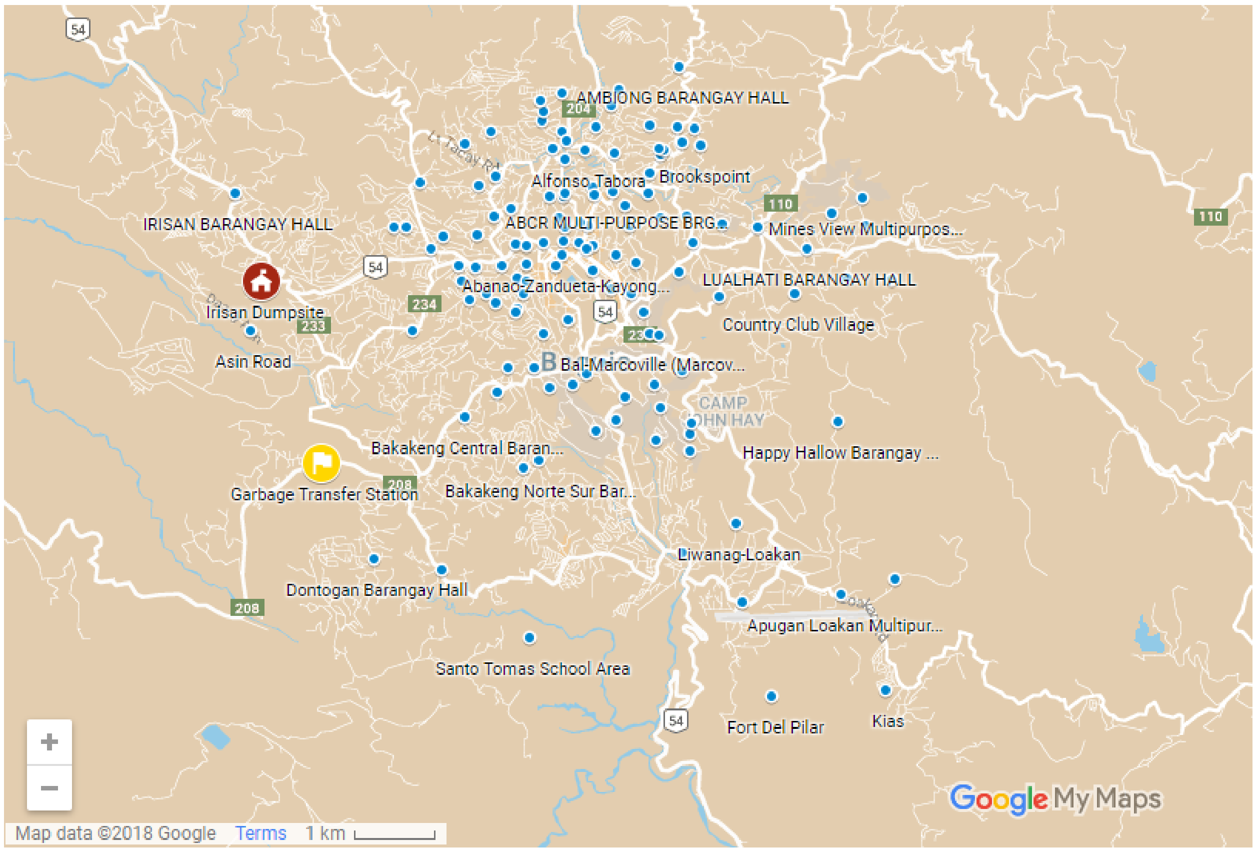

Figure 16: A Map of the Locations of the Barangays

where

$$v_i = \begin{cases} v_0 & \text{is the Depot} \\ v_1, v_2, \ldots, v_n & \text{are the Collection Sites} \\ v_{n+1} & \text{is the Disposal Site} \end{cases}$$

Each vertex $v_i \in V$ is associated with a demand q_i equivalent to the amount of garbage to be collected in cubic meters.

$$q_i = \begin{cases} q_0 = 0 \text{ m}^3 \\ q_1, q_2, \ldots, q_n \in \mathbb{Z}, \text{ specifically } \in (0, Q) \text{ m}^3 \\ q_{n+1} = 0 \text{ m}^3 \end{cases}$$

wherein,

$$\sum_{i=1}^{n} q_i = W$$

where W is the total amount of garbage generated by all collection sites and Q is the maximum carrying capacity of any vehicle defined below.

The set of edges

$$E = \{(v_i, v_j) | v_i, v_j \in V, \ i, j \in 0, 1, 2 \ldots, n+1\}$$

The edge $(v_i, v_j) \in E$ connects an arbitrary pair of vertices v_i, v_j in graph G.

Each edge $(v_i, v_j) \in E$ is associated to a distance $d_{i,j}$ in kilometers. In this case, we use the exact distance in kilometers given by Google Maps© Distance API.

Let $K = \{k_i\}, i \in 1, 2, 3, \ldots, m$ be the set of waste collection vehicles. There are a fixed number of m vehicles.

Let Q be the maximum carrying capacity of any vehicle $k \in K$. This is the maximum amount of garbage that can be collected and carried by a vehicle along it's path.

Let A_{ijl} be the accumulated amount collected by vehicle $k_l \in K$ when moving between v_i and v_j where $l \in 1, 2, \ldots, m$ and $v_i, v_j \in V, \ i, j \in 0, 1, 2 \ldots, n+1$.

The decision variables of the model depend on the vehicle capacity Q and the waste quantity at the next waste collection site it visits. These are modeled as follows:

$$X_{i,j,l} = \begin{cases} 1, & \text{if vehicle } k_l \text{ can travel from vertex } v_i \text{ to } v_j \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$Y_{i,l} = \begin{cases} 1, & \text{if vertex } v_i \text{ is visited by vehicle } k_l \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Our aim is to minimize the total travel distance F while maximizing waste collected. Since our waste collected is in cubic meters, we can obtain an equivalent amount in kilometers by the formula $\sqrt[3]{\text{Amount } \cancel{m}^3} \times \frac{1km}{1000\cancel{m}} = \frac{\text{Amount}}{1000} km$. Hence, our objective function is represented by the equation

$$F = \min \left(\sum_{l=1}^{m} \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} X_{i,j,l} \cdot d_{i,j} + \frac{\alpha}{1000} \cdot \sqrt[3]{W - \left(\sum_{l=1}^{m} \sum_{i=1}^{n} Y_{i,l} \cdot q_i \right)} \right) \tag{4}$$

Where α is some constant which determines the magnitude of the effect imposed upon by uncollected waste. $\alpha \geq 1000$ so that the effect is substantial enough to give feasible routes with much more collected waste to be converged upon.

In order to make satisfy our assumptions, we subject our objective function following constraints:

$$\sum_{i=1}^{n}\sum_{l=1}^{m} X_{i,j,l} = 1, \qquad \forall j = 1, 2, \ldots, n \tag{5}$$

$$\sum_{i=1}^{n} Y_{i,l} = \sum_{i=1}^{n} X_{i,j,l}, \qquad \forall l = 1, 2, \ldots, m; j = 1, 2, \ldots, n \tag{6}$$

$$\sum_{j=1}^{n+1} X_{0,j,l} = 1, \qquad \forall l = 1, 2, \ldots, m \tag{7}$$

$$\sum_{i=1}^{n+1} X_{i,0,l} = 1, \qquad \forall l = 1, 2, \ldots, m \tag{8}$$

$$\sum_{j=1}^{n+1} A_{0,j,l} = 0, \qquad \forall l = 1, 2, \ldots, m \tag{9}$$

$$\sum_{i=1}^{n+1} A_{i,0,l} = 0, \qquad \forall l = 1, 2, \ldots, m \tag{10}$$

$$A_{i,j,l} \leq Q, \qquad \forall l = 1, 2, \ldots, m; i, j = 0, 1, \ldots, n \tag{11}$$

$$\sum_{j=1}^{n} A_{j,h,l} - A_{i,j,l} = \sum_{j=1}^{n} Y_{il} \cdot q_j, \qquad \forall l = 1, 2, \ldots, m; i, h = 0, 1, \ldots, n \tag{12}$$

$$d_{i,j} = d_{j,i}, \qquad \forall i = 0, 1, \ldots, n; j = 0, 1, \ldots, n \tag{13}$$

$$X_{i,j,l} \in 1, 0 \tag{14}$$

$$Y_{i,l} \in 1, 0 \tag{15}$$

Constraint (5) specifies that collection site v_i is visited by not more than one vehicle k_l and (6) specifies that a collection site v_i is in the route of vehicle k_l. Since the values of each

X_{ijl} is 1 if vehicle k_l moves from vertex v_i to v_j and 0 otherwise, then if we get the sum of the values, we will know how many times v_i is visited by all vehicles. However, we assumed that we only visit each collection site once, hence, the sum must be equal to one.

Constraints (7) and (8) imposes that each vehicle $k_l \in K$ must start and end at the depot. Constraints (9) and (10) imposes that each vehicle $k_l \in K$ must have no accumulated waste before leaving and returning to the depot.

Constraint (11) imposes that the accumulated amount of any vehicle $k_l \in K$ traveling between any pair of vertices v_i and v_j must be less than the maximum capacity.

Constraint (12) imposes that the vehicle k_l completely collects all waste when it visits vertex v_j.

Constraint (13) shows that the distance of two vertices traveled back and forth are the same since we assumed that graph G is an undirected connected graph which means that it is symmetric as well.

Constraints (14) and (15) define the domain of the decision variables.

6 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. [25, 26] 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. The flowchart for the algorithm is shown on figure 17

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 $F(x_{id})$, $i \in (1, 2, 3, \ldots, N)$

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 $i \in (1, 2, 3, \ldots, N)$. We take note that each dimension of the search space is usually bounded or are in intervals $[a_j, b_j]$, $j \in (1, 2, 3, \ldots, d)$. 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.

6.1 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[27], Heppner and Grenander [28] have tried to simulate movements of birds and fish using the

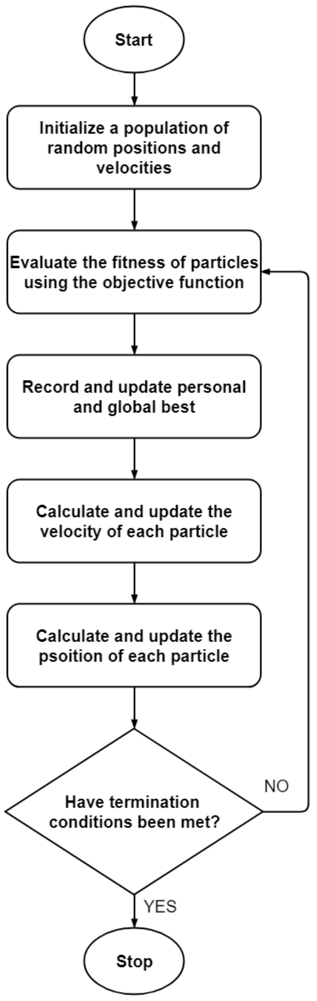
Start
Initialize a population of random positions and velocities
Evaluate the fitness of particles using the objective function
Record and update personal and global best
Calculate and update the velocity of each particle
Calculate and update the psoition of each particle
Have termination conditions been met?
NO
YES
Stop

Figure 17: Flowchart of the PSO Algorithm

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[28], 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:

$$V[i][d] = c1 * rand() * (pbest[i][d] - present[i][d]) + c2 * rand() * pbest[gbest][d] - present[i][d]) \tag{16}$$

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, $x \in (1, 2, 3, \ldots, n)$

Eberhart and Kennedy [25] 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.

6.2 Further Developments

Eberhart and Shi[29] 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[29] further improved the original algorithm proposed by Eberhart and Kennedy[25] by introducing inertia weight. Inertia weight is responsible for balancing global and local exploration. The new velocity equation becomes

$$v_{id} = v_{id} * w_{id} + c_1 * rand() * (pbest_{id} - x_{id}) + c_2 * rand() * (pbest_{gd} - x_{id}) \tag{17}$$

where the new variable w_{id} is the inertia weight. Eberhart and Shi[29] 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.

6.2.1 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.[30] 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[31]. 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 $\varphi > 4$.

6.2.2 Chaos Search

Chaos is a characteristic of non-liner system that includes infinite unstable periodic motions and depends on initial conditions.[32] 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 [33],

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

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 [33] 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 [34]:

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

Although it is not very different from the equation of Kennedy et.al.[25], 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.

6.2.3 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[35]. Sun et.al.[35] 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.[35, 36]
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.

7 GA

Genetic Algorithm (GA) is an evolutionary algorithm developed by John Holland et. al.[37] 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.

GA was developed by John Holland[37] 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.

7.1 Components of GA

The basic algorithm for GA is shown below. A flowchart of the algorithm is also shown on figure 18.

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 $i \in (1, 2, 3, \ldots, N)$ and $d = (1, 2, 3, \ldots, D)$
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)

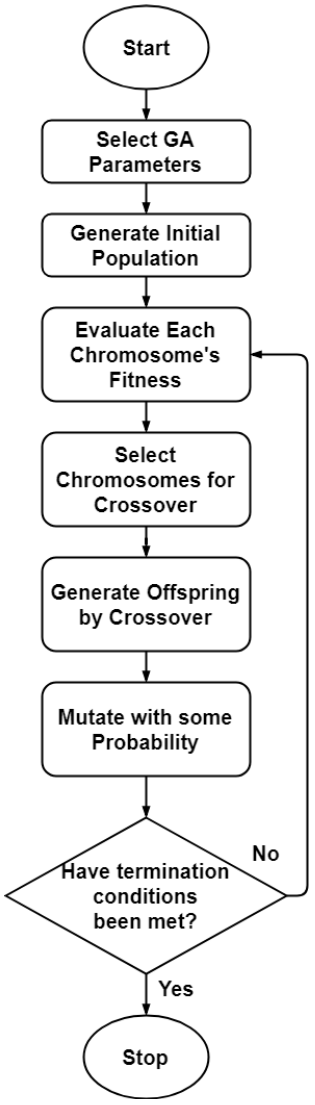
Start
Select GA Parameters
Generate Initial Population
Evaluate Each Chromosome's Fitness
Select Chromosomes for Crossover
Generate Offspring by Crossover
Mutate with some Probability
Have termination conditions been met?
No
Yes
Stop

Figure 18: Flowchart of the GA Algorithm

   (c) (Mutation) With a mutation probability, mutate new offspring at some positions $x_d, d \in (1, 2, 3, \ldots, D)$
   (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.

7.1.1 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.[38] This is because the population size becomes too big for the solution space, or the number of computations needed becomes too large and redundant.

7.1.2 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.[39] 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.

7.1.3 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 19. 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 20.

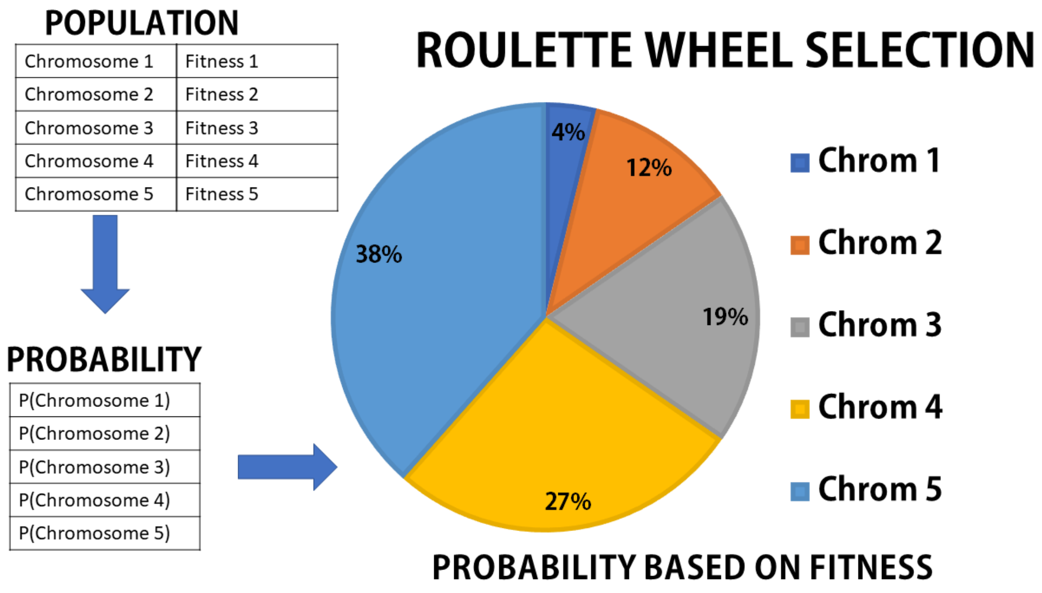

Figure 19: Roulette Wheel Selection

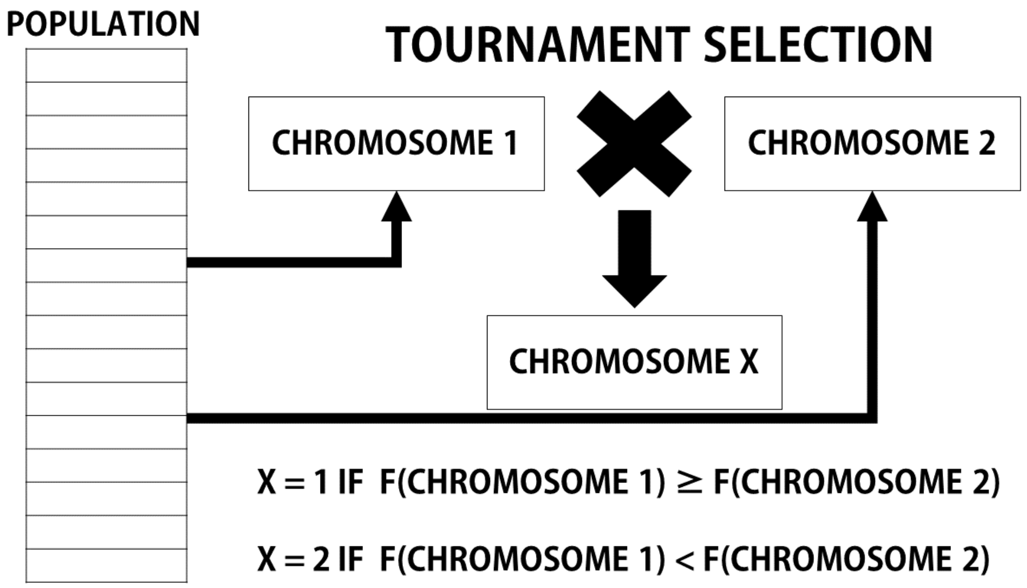
POPULATION
TOURNAMENT SELECTION
CHROMOSOME 1
CHROMOSOME 2
CHROMOSOME X
X = 1 IF F(CHROMOSOME 1) ≥ F(CHROMOSOME 2)
X = 2 IF F(CHROMOSOME 1) < F(CHROMOSOME 2)

Figure 20: Simple Elimination Selection

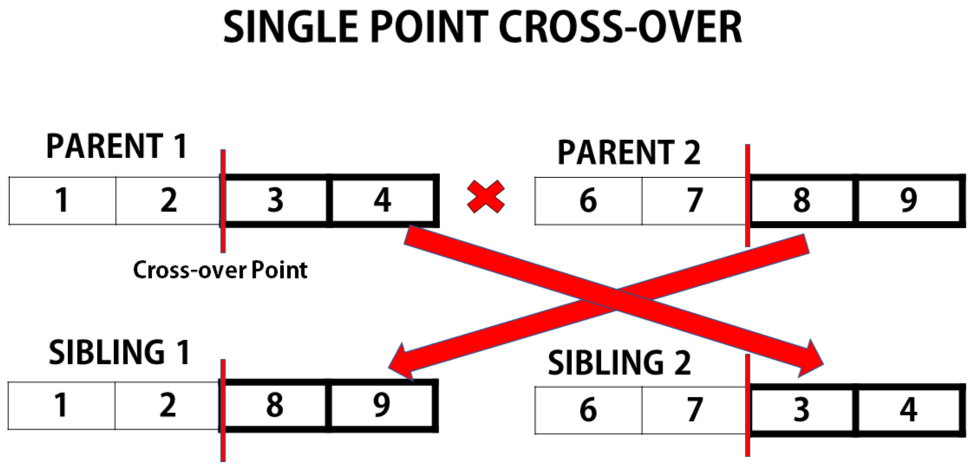

Figure 21: Singe Point Cross-Over

7.1.4 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 21.

7.1.5 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 22. If the chromosomes are

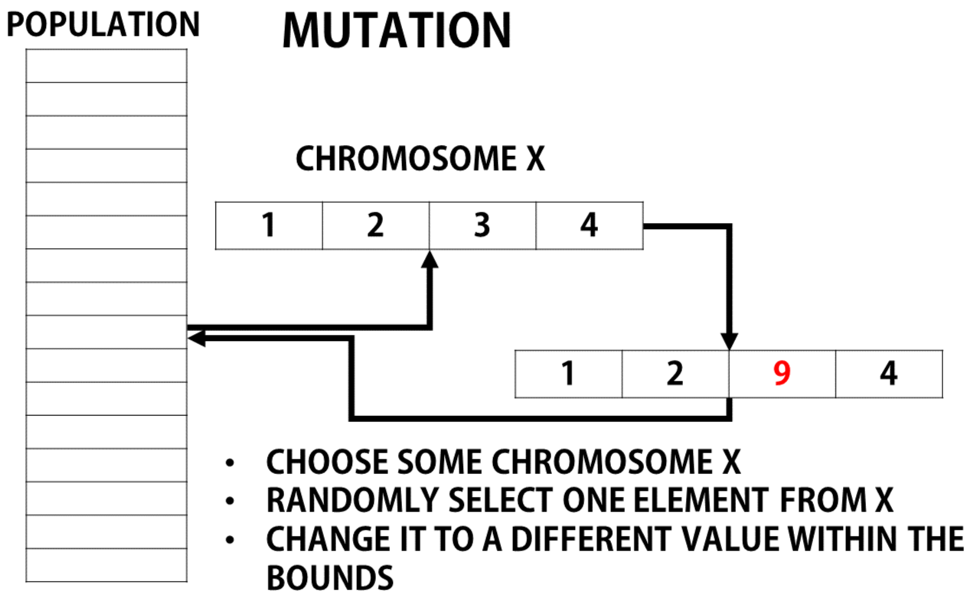
POPULATION
MUTATION
CHROMOSOME X
1
2
3
4
1
2
9
4
• CHOOSE SOME CHROMOSOME X
• RANDOMLY SELECT ONE ELEMENT FROM X
• CHANGE IT TO A DIFFERENT VALUE WITHIN THE BOUNDS

Figure 22: Mutation in GA

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.

7.1.6 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.

7.1.7 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.

7.1.8 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.

8 PSO-GA Approach

PSO-GA is a hybrid of PSO and GA. Harish Garg has proposed a PSO-GA[16] 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[16] 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.

8.1 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

$$GA_{Num} = GA_{NumMax} - (\frac{PSO_{CurrIt}}{PSO_{MaxIt}})^{\gamma} \times (GA_{NumMax} - GA_{NumMin}) \tag{18}$$

$$GA_{PopSize} = GA_{MinPopSize} + (\frac{PSO_{CurrIt}}{PSO_{MaxIt}})^{\gamma} \times (GA_{MaxPopSize} - GA_{MinPopSize}) \tag{19}$$

$$GA_{MaxItr} = GA_{MinItr} + (\frac{PSO_{CurrIt}}{PSO_{MaxIt}})^{\beta} \times (GA_{MaxItr} - GA_{MinItr}) \tag{20}$$

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 ($i \in 1, 2, 3, \ldots, PSO_{PopNum}$)
6. Update each particle's velocity and position with the equations,

$$w = w_{max} - (w_{max} - w_{min}) \times (\frac{PSO_{CurrIt}}{PSO_{MaxIt}}) \tag{21}$$

$$v_i = v_i \times w + c_1 \times rand() \times (pbest_i - PSOx_i) + c_2 \times rand() \times (pbest_g - PSOx_i) \tag{22}$$

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

$$PSOx_i = PSOx_i + v_i \tag{23}$$

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 $PSOx_s$

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

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 (18), (19) and (20)
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

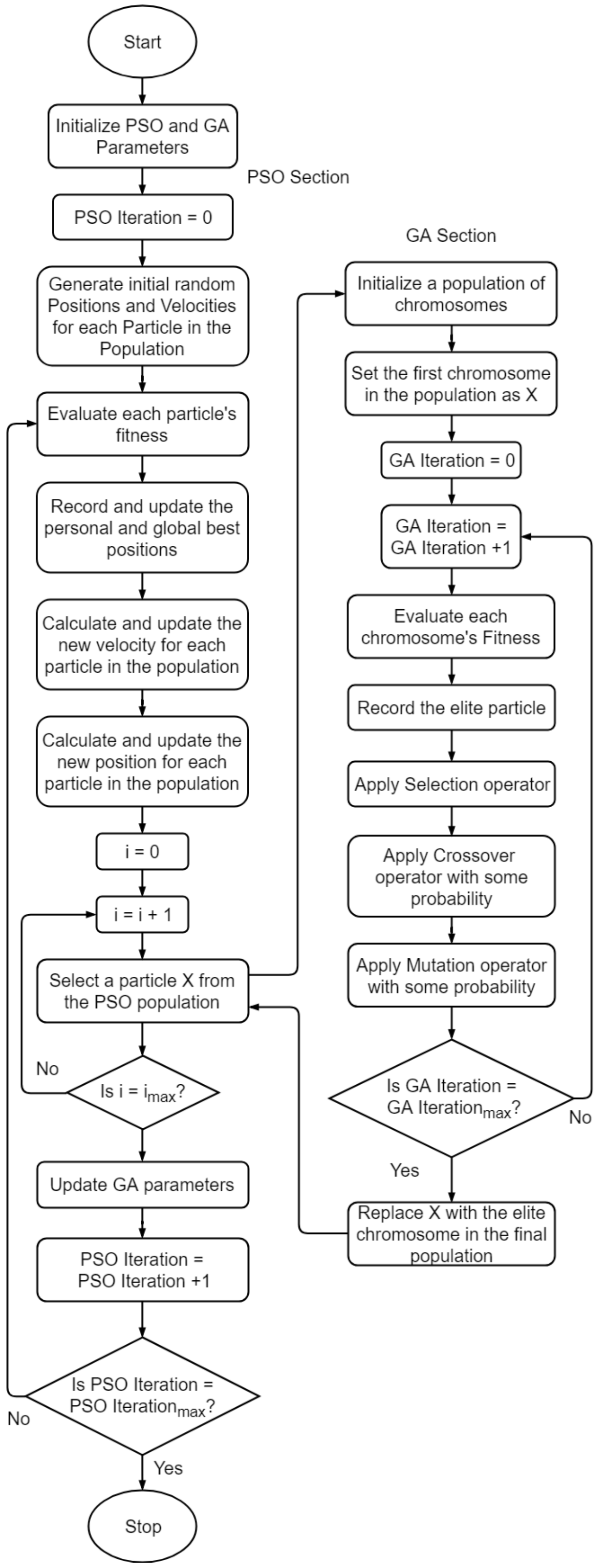
Start
Initialize PSO and GA Parameters
PSO Section
PSO Iteration = 0
Generate initial random Positions and Velocities for each Particle in the Population
Evaluate each particle's fitness
Record and update the personal and global best positions
Calculate and update the new velocity for each particle in the population
Calculate and update the new position for each particle in the population
i = 0
i = i + 1
Select a particle X from the PSO population
No
Is i = i_{max}?
Update GA parameters
PSO Iteration = PSO Iteration +1
Is PSO Iteration = PSO $Iteration_{max}$?
No
Yes
Stop
GA Section
Initialize a population of chromosomes
Set the first chromosome in the population as X
GA Iteration = 0
GA Iteration = GA Iteration +1
Evaluate each chromosome's Fitness
Record the elite particle
Apply Selection operator
Apply Crossover operator with some probability
Apply Mutation operator with some probability
Is GA Iteration = GA $Iteration_{max}$?
No
Yes
Replace X with the elite chromosome in the final population

Figure 23: Flowchart of PSO GA Algorithm

The flowchart of the algorithm is shown on figure 23

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 (18), (19) and (20).

Judging from the equations 18, 19 and 20, GA_{Num} will initially be GA_{NumMax} and slowly become GA_{NumMin} as the number of iterations increases. This is because the fraction PSO_{CurrIt}/PSO_{MaxIt} is raised to γ which is a positive whole number as given by Garg[16], 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 $w = w_{max} - (w_{max} - w_{min})(PSO_{CurrIt}/PSO_{MaxIt})$, it is most of the time a fraction. Garg[16] started $w = 0.9$ initially then becoming $w = 0.4$ as the number of iterations increases.

9 Test Case

We first test if the PSO-GA algorithm works. Liu et.al.[24] 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 2. Vertex 0 is the depot while the rest are the 8 customers. The distances assigned to the edges are given in table 3. 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, $K = \{1, 2, 3\}$.

Table 2: 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 3: 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

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

$$\text{Total Cost } = \text{Cost}_{\text{distance}} + \text{Cost}_{\text{waiting}} + \text{Cost}_{\text{late}} \tag{24}$$

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:

$$
\begin{aligned}
arrival_{0,1} &= 0 \\
&P_{E_{0,1}} = \max\{0, (a_0 - arrival_{0,1})\} = \max\{0, 0\} = 0 \\
&P_{L_{0,1}} = \max\{0, (arrival_{0,1} - b_0)\} = \max\{0, -14\} = 0 \\
arrival_{3,1} &= \max\{a_0, arrival_{0,1}\} + d_{(0,3)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_0 \\
&= 0 + \frac{75}{50} + 0 = 1.5 \text{ hours} \\
&P_{E_{3,1}} = \max\{0, (a_3 - arrival_{3,1})\} = \max\{0, -0.5\} = 0 \\
&P_{L_{3,1}} = \max\{0, (arrival_{3,1} - b_3)\} = \max\{0, -0.5\} = 0 \\
arrival_{1,1} &= \max\{a_3, arrival_{3,1}\} + d_{(3,1)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_3 \\
&= 1.5 + \frac{40}{50} + 1 = 3.3 \text{ hours} \\
&P_{E_{1,1}} = \max\{0, (a_1 - arrival_{1,1})\} = \max\{0, -2.3\} = 0 \\
&P_{L_{1,1}} = \max\{0, (arrival_{1,1} - b_1)\} = \max\{0, -0.7\} = 0 \\
arrival_{2,1} &= \max\{a_1, arrival_{1,1}\} + d_{(1,2)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_1 \\
&= 3.3 + \frac{65}{50} + 1 = 5.6 \text{ hours} \\
&P_{E_{2,1}} = \max\{0, (a_2 - arrival_{2,1})\} = \max\{0, -1.6\} = 0 \\
&P_{L_{2,1}} = \max\{0, (arrival_{2,1} - b_2)\} = \max\{0, -0.4\} = 0 \\
arrival_{0,1} &= \max\{a_2, arrival_{2,1}\} + d_{(2,0)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_2 \\
&= 5.6 + \frac{60}{50} + 2 = 7.0 \text{ hours} \\
&P_{E_{0,1}} = \max\{0, (a_0 - arrival_{0,1})\} = \max\{0, -7.0\} = 0 \\
&P_{L_{0,1}} = \max\{0, (arrival_{0,1} - b_0)\} = \max\{0, -7.0\} = 0
\end{aligned}
$$

- Vehicle 2:

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

- Vehicle 3:

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

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:

CPU = Intel i5-6200U 2.3 GHz

RAM = 16 Gb

OS = Windows 10 Home 2017

The results are presented in the following table:

Table 4: Summary of Results (8 Customers, 3 Vehicles, PSO-GA)

Maximum Iteration	Best Value Found	Worst Value Found	Number of Converged Runs	Number of Successful Converged Runs	Average Running Time (s)	Standard Deviation
200	910	1425	3/10	1/3	104.094591	144.076561
400	910	1370	4/10	1/4	189.477791	131.471374
800	910	1255	8/10	1/8	377.513529	87.311702
1200	910	1100	4/10	1/4	588.434658	63.674519
1600	910	1070	7/10	3/7	742.915872	68.315038

As we can see, the algorithm found the optimal solution, which has the cost of 910. However, most the tests only had 1 successful converged runs. A successful converged run is where the population converged on the optimal solution. There were other converged runs however, the population converged at sub-optimal locations. It is also observed that there is a large standard deviation, this means that the solutions found in all runs vastly varies but the variation decreases as maximum iteration increases.

10 Simple Example

10.1 TSP with Time Windows

We try a simple example wherein we have 3 customers, 1 depot and 1 vehicle. Time is measured in hours and distance in yards. Node 0 is the depot and the rest of the nodes are the 3 customers respectively. Note that this is a TSP problem since there is only one vehicle but it involves time windows.
Given the distances of each node $(i, j) \in V$ Given the service time, waste to collect, and time windows at each node $i \in V$. Note that vertex 0 is the depot and nodes $1-3$ are customers.

Table 5: Distances from Point to Point (Yards)

$d_{(i,j)}$	**0**	**1**	**2**	**3**
0	**0**	**120**	**80**	**50**
1	**120**	**0**	**70**	**100**
2	**80**	**70**	**0**	**30**
3	**50**	**100**	**30**	**0**

Table 6: Node Characteristics

Vertices (i)	**0**	**1**	**2**	**3**
Load (q_i)	**0**	**4**	**2**	**2**
Service Time (s_i)	**0**	**1**	**2**	**1**
Time Window $[a_i, b_i]$	**[0,12]**	**[1,4]**	**[5,6]**	**[2,5]**

We consider 1 vehicle. Hence, there are 3! = 6 possible permutations from which there is one known solution whose fitness value is 400:

- Vehicle Route: $(0 \rightarrow 1 \rightarrow 3 \rightarrow 2 \rightarrow 0)$

Note that we are still using the same cost function (24). We solve the fitness of this route by first calculating the total distance

$$\begin{aligned}
\text{Total Distance} &= d_{(0,1)} + d_{(1,3)} + d_{(3,2)} + d_{(2,0)} \\
\text{Total Distance} &= 120 + 100 + 30 + 80 \\
\text{Total Distance} &= 330
\end{aligned}$$

Now we calculate the cost of being early and late.

- Vehicle Route:

$$\begin{aligned}
arrival_0 &= 0\\
&\quad P_{E_0} = \max\{0,(a_0 - arrival_0)\} = \max\{0,0\} = 0\\
&\quad P_{L_0} = \max\{0,(arrival_0 - b_0)\} = \max\{0,-12\} = 0\\
arrival_1 &= \max\{a_0, arrival_0\} + d_{(0,1)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_0\\
&= 0 + \frac{120}{50} + 0 = 2.4 \text{ hours}\\
&\quad P_{E_1} = \max\{0,(a_1 - arrival_1)\} = \max\{0,-1.4\} = 0\\
&\quad P_{L_1} = \max\{0,(arrival_1 - b_1)\} = \max\{0,-1.6\} = 0\\
arrival_3 &= \max\{a_1, arrival_1\} + d_{(1,3)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_1\\
&= 2.4 + \frac{100}{50} + 1 = 5.4 \text{ hours}\\
&\quad P_{E_3} = \max\{0,(a_3 - arrival_3)\} = \max\{0,-3.4\} = 0\\
&\quad P_{L_3} = \max\{0,(arrival_3 - b_3)\} = \max\{0,0.4\} = 0.4\\
arrival_2 &= \max\{a_3, arrival_3\} + d_{(3,2)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_3\\
&= 5.4 + \frac{30}{50} + 1 = 7 \text{ hours}\\
&\quad P_{E_2} = \max\{0,(a_2 - arrival_2)\} = \max\{0,-2\} = 0\\
&\quad P_{L_2} = \max\{0,(arrival_2 - b_2)\} = \max\{0,1\} = 1\\
arrival_0 &= \max\{a_2, arrival_2\} + d_{(2,0)} \times \frac{1 \text{ hour}}{50 \text{ yards}} + s_2\\
&= 7 + \frac{80}{50} + 2 = 10.6 \text{ hours}\\
&\quad P_{E_0} = \max\{0,(a_0 - arrival_0)\} = \max\{0,-10.6\} = 0\\
&\quad P_{L_0} = \max\{0,(arrival_0 - b_0)\} = \max\{0,-1.4\} = 0
\end{aligned}$$

- Total Time Early:

$$\begin{aligned}
P_E &= 0 + 0 + 0 + 0 + 0\\
&= 0
\end{aligned}$$

- Total Time Late:

$$\begin{aligned}
P_E &= 0 + 0 + 0.4 + 1 + 0\\
&= 1.4
\end{aligned}$$

Hence the total cost of being early and late is $1.4 \cancel{\text{hour}} \times \frac{50 \text{ yards}}{1 \cancel{\text{hour}}} = 70$ yards. Therefore the total cost in yards is $330 + 70 = 400$. The possible permutations and their respective fitness values are given as follows:

- Vehicle Route: $(0 \to 3 \to 2 \to 1 \to 0) = 610$
- Vehicle Route: $(0 \to 2 \to 3 \to 1 \to 0) = 960$
- Vehicle Route: $(0 \to 3 \to 1 \to 2 \to 0) = 470$
- Vehicle Route: $(0 \to 2 \to 1 \to 3 \to 0) = 1010$
- Vehicle Route: $(0 \to 1 \to 2 \to 3 \to 0) = 410$
- Vehicle Route: $(0 \to 1 \to 3 \to 2 \to 0) = 400$

We first test the PSO algorithm using the following factors:

- PSO Population Size = 3, 6, 15, 40
- PSO Maximum Iterations = PSO Population Size $\times$ 5
- Cognitive and Social factors $c_1 = 1.5$, $c_2 = 1.5$
- Initial and final inertia weight $w_i = 0.9$ $w_f = 0.4$

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

CPU = Intel i5-6200U 2.3 GHz

RAM = 16 Gb

OS = Windows 10 Home 2017

The results are presented in the following table:

Table 7: Summary of Results (3 Customers, 1 Vehicle, PSO)

Population Size	Maximum Iteration	Best Value Found	Worst Value Found	Number of Converged Runs	Number of Successful Converged Runs	Average Running Time (s)	Standard Deviation
3	15	400	610	15/20	11/15	0.005620	51.121630
6	30	400	470	19/20	17/19	0.012434	15.694451
15	75	400	1255	19/20	19/19	0.072328	0.000000
40	200	400	1100	18/20	18/18	0.563054	0.000000

The results show that as the population size increases, the solutions obtained become more stable. The number of successful converged runs (runs that converged to the optimal

solution) for all test are more than 50%. The standard deviation tells us that there are no deviations for the solutions obtained on higher population sizes.
Now we test the PSO-GA algorithm for the same problem. The PSO-GA algorithm was used using the following factors:

- PSO Population Size = 3, 6, 15, 40
- PSO Maximum Iterations = PSO Population Size $\times$ 5
- 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 Minimum Replacement = 1
- GA Maximum Replacement = 1 (for Pop. Size = 3), 3 (for the rest)
- GA Initial Population Size = 10
- GA Final Population Size = 5
- GA Minimum Iterations = 10
- GA Maximum Iterations = 15

Table 8: Summary of Results (3 Customers, 1 Vehicle, PSO-GA)

Population Size	Maximum Iteration	Best Value Found	Worst Value Found	Number of Converged Runs	Number of Successful Converged Runs	Average Running Time (s)	Standard Deviation
3	15	400	960	19/20	10/19	0.123629	124.260973
6	30	400	470	20/20	19/20	0.561680	15.652476
15	75	400	400	20/20	20/20	1.516562	0.000000
40	200	400	400	19/20	19/19	4.279840	0.000000

The results show that as the population size increases, the solution becomes more stable. Moreover, PSO and PSO-GA have almost the same results with a few differences. The major difference is in the average running time. PSO-GA is expected to run slower because it is a hybrid where more computations are needed.

10.2 2 Vehicles

The previous tests resulted in almost the same results, now we test the same 3 nodes but this time, we now consider 2 vehicles and reduce each vehicle's capacity to 4. Hence, there are 4! = 24 possible permutations from which there are two known solutions whose fitness is given as 520:

- Vehicle 1: $(0 \rightarrow 1 \rightarrow 0)$
- Vehicle 2: $(0 \rightarrow 3 \rightarrow 2 \rightarrow 0)$

  and

- Vehicle 1: $(0 \rightarrow 3 \rightarrow 2 \rightarrow 0)$
- Vehicle 2: $(0 \rightarrow 1 \rightarrow 0)$

This time we test the PSO algorithm first using the following factors:

- PSO Population Size = 7, 15, 24, 40
- PSO Maximum Iterations = PSO Population Size × 5
- Cognitive and Social factors $c_1 = 1.5$, $c_2 = 1.5$
- Initial and final inertia weight $w_i = 0.9$ $w_f = 0.4$

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

CPU = Intel i5-6200U 2.3 GHz

RAM = 16 Gb

OS = Windows 10 Home 2017

The results are presented in the following table:

Table 9: Summary of Results (3 Customers, 2 Vehicles, PSO)

Population Size	Maximum Iteration	Best Value Found	Worst Value Found	Number of Converged Runs	Number of Successful Converged Runs	Average Running Time (s)	Standard Deviation
7	35	520	650	14/20	13/14	0.027327	40.013156
15	75	520	520	16/20	16/16	0.101834	0.000000
24	120	520	520	17/20	17/17	0.221712	0.000000
40	200	520	520	13/20	13/13	0.718284	0.000000

The results show that as the population size increases, the solution becomes more stable however, if the population size exceeds some point, there are less converged runs. This is

Table 10: Summary of Results (3 Customers, 2 Vehicles, PSO-GA)

Population Size	Maximum Iteration	Best Value Found	Worst Value Found	Number of Converged Runs	Number of Successful Converged Runs	Average Running Time (s)	Standard Deviation
7	35	520	520	15/20	15/15	1.989956	0.000000
15	75	520	520	17/20	17/17	4.583488	0.000000
24	120	520	520	17/20	17/17	7.187364	0.000000
40	200	520	520	12/20	12/12	14.240547	0.000000

possibly because there are more members in the population for convergence to occur before the maximum iteration is reached. In more than 50% of the runs, the population successfully converged to the optimal solution.
Now we test the PSO-GA algorithm for the same problem. The PSO-GA algorithm was used using the following factors:

- PSO Population Size = 7, 15, 24, 40
- PSO Maximum Iterations = PSO Population Size $\times$ 5
- 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 Minimum Taken = 1
- GA Maximum Taken = 5
- GA Initial Population Size = 10
- GA Final Population Size = 5
- GA Minimum Iterations = 10
- GA Maximum Iterations = 15

The results show that as the population size increases, the solution becomes more stable however, at some point, the number of converged runs become less. This is so maybe because there are more members in the population that need to converge and that the maximum iteration is small. If we take a look at the results of both algorithms, PSO-GA performed better and is more consistent with the number of iterations. All of PSO-GA's converged runs are all successful ones, this means that for all times the population converged, it obtained the optimal solution. If we take the average running time into consideration, PSO-GA is less efficient but this is normal because the PSO-GA algorithm has more complexity and computations.

References

[1] D. Hoornweg and B. Perinaz, "What a waste: a global review of solid waste management," vol. 15, pp. 87–88, 01 2012.

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

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

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

[5] 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.*

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

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

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

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

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

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

[22] 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.

[23] 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.

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

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

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

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

[28] F. Heppner and G. U.

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

[30] 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.

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

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

[38] M. Obitko, “Intoduction to genetic algorithms.”

[39] J. Carr, “An introduction to genetic algorithms,” 2014.