Solving Optimization Problems using Evolutionary Algorithms

Garv Sudhir Nair, Michael Lu Han Xien, Anjali Radha Krishna, Liew Tze Chen, Liew Hsien Loong

Department of Computing and Information Systems, Sunway University
Subang Jaya, Malaysia

19073535@imail.sunway.edu.my

18081588@imail.sunway.edu.my

16009847@imail.sunway.edu.my

18032730@imail.sunway.edu.my

18030221@imail.sunway.edu.my

***Abstract* – This document contains the use of a Genetic Algorithm and Particle Swarm Optimization, which are both Evolutionary Algorithms, to find the best results for each scenario presented. The optimal seating arrangement based on happiness values being determined through the genetic algorithm and the optimal time to leave the premises is determined through the particle swarm optimization. The results of each experiment are analyzed and discussed to justify the choices made during testing and result selection.**

***Keywords* – Genetic Algorithm, Evolutionary Algorithms, Computational Intelligence, Particle Swarm Optimization**

I. Introduction

In the field of Artificial Intelligence, computational methodologies are employed to study and solve real world problems and complex issues which cannot be answered through traditional means using human logic and reasoning. This logic and reasoning is carried out by a computer instead of a person. Computational Intelligence introduces us to concepts widely used today that are derivatives of work done by humans and nature - such as the concept of artificial neural networks that mimic biology in its methods as well as evolutionary computing that introduces a computed take on natural selection and learning methods.

The field of computational intelligence comes with its vast array of algorithms and practices employing the principles of the human way of thinking. Evolutionary algorithms are commonly used to study all types of problems as they are well equipped to process complex problems even without aid using fitness approximations. The genetic algorithm is one commonly used amongst others and is further explored in this document. The algorithm is employed to study the happiness of each person in relation to their seating position and to determine the most ideal seating position for a set group of people.

The next problem at hand would be determining the ideal time for a person to move out of their accommodation based on factors such as the renovation level of the accommodation, the time and day of moving, the price to be paid when leaving, and other relevant factors. The Particle Swarm Optimization is chosen to factor in these values and determine the optimal time. The algorithm is used to search for the best solution over an array of points when the next best method is generally an exhaustive search and inspection of each point. Upon several iterations, the algorithm can produce a viable solution.

II. Genetic Algorithm

A. Algorithm explanation

Genetic Algorithm (GA) is a commonly known as a search heuristic inspired by Charles Darwin's natural

evolution theory based on the principles of natural selections and genetics [1]. It is an optimization process of which the purpose is to identify the chromosome with the best combination of genes. This algorithm reflects the process of natural selection whereby only the fittest individual in the population will be chosen for reproduction to produce the offspring of the next generation. There are a few phases to be considered in the Genetic Algorithm which are the initialization of population, reproduction, mutation, and replacement of the population.

During the initialization of the population, the fitness function is used for evaluating the fitness of each chromosome. Then, a population will be created with an n number of chromosomes where in this chromosome will be the first ancestor. The value of n is important as it will determine whether the population will have a higher or lower diversity. In other words, the relationship between n chromosomes and diversity are directly proportional. However, the complexity will also increase with n chromosomes due to the increase in diversity.

Next, the chromosomes will interact with each other and reproduce with the process called crossover. Crossovers come under various categories some of which are sexual, asexual, and multi recombination. To create offspring, this operator exchanges genetic information between two parents. This is performed on parent pairs that are randomly chosen to generate the child population with the size same as the parent population [2]. Besides, the process of mutation will not necessarily occur in the children, but it does have the probability of occurring.

The purpose of mutation is to create new genetic information or different combinations by flipping random bits in the chromosome. By having new genetic information, the diversity of the population will be increased as well. With the new group of parents and children, the children with good fitness will be selected to replace the parents. The parents with good fitness will also remain in the population whereas parents and children with bad fitness will be discarded from the population. The selection can be done with various different methods, some of which include random selection, proportional selection, tournament selection, rank-based election, and elitism selection.

When the fittest parents and children are selected as the new population, the process will repeat back to the reproduction process and go on until there is no improvement in terms of fitness and no change of chromosomes in a few generations. Examples of termination criteria are as shown below:

- No improvement in the population for a Y iterations
- Absolute number of generations is achieved
- A certain pre-defined value is reached from the objective function value

B. Implementation

1) Search space - The search space refers to as a set or domain in which an algorithm performs searches. The space is a well-defined and finite data structure. It can be said that it is a set of feasible valid moves. In this case, the chromosome in the search space is locating the best seating plan arrangement for individual A, B, C, D, and E. Each of the individuals are indicated with a happiness scale range from -100 to 100 with respect to one another.

2) Fitness Function - The fitness function is a performance measure to determine how near a given solution is to the optimal solution for the situation at hand. The optimal solution would be obtaining the highest happiness score in total among the individuals in the seating plan arrangement mentioned above. For example, if the seating arrangement is C-D-E, the happiness of each person is calculated as below:

$$\begin{aligned} Happiness\ of\ C &= Happiness\ of\ C\ sitting\ next\ to\ D \\ Happiness\ of\ D &= Happiness\ of\ D\ sitting\ next\ to\ C \\ &\quad + Happiness\ of\ D\ sitting\ next\ to\ E \\ Happiness\ of\ E &= Happiness\ of\ E\ sitting\ next\ to\ D \end{aligned}$$

All happiness values are stored within an array to resemble the happiness matrix provided in the problem statement.

```
People (p) = [
          A: [0, 20, 20, 30, 25],
          B: [20, 0, 50, 20, 5],
          C: [10, 10, 0, 100, 10],
          D: [50, 5, 10, 0, -5],
          E: [-50, -30, -100, -10, 0]
        ]
```

After a combination is passed to the function, it groups the combination into sets of pairs and iterates by using

the seat values to reference our happiness matrix. An example of such would be,

$$seats\ (s) = [1, 2, 3, 4, 5]$$

Where each numeral represents the label of a person, alternatively the seating arrangement could also be written as $[A, B, C, D, E]$. By splitting these into pairs, one can surmise the happiness of the whole arrangement to equate to the following,

$$Total\ Happiness = p[1,2] + p[2,3] + \ \ldots + p[i, j]$$

Where p refers to the previously defined happiness matrix.

By further simplifying this formula one can formulate:

$$Total\ Happiness = \sum_{k=1}^{n-1} p[s_k, s_{k+1}] + p[\ s_{k+1},\ s_k]$$

3) Crossover Function – The crossover function is primarily used to introduce the concept of breeding into the algorithm. Unfortunately, however, making use of traditional crossover methods would result in duplicate values appearing within the combination of seats, i.e. [A, B, C, D, A] has two individuals named A. This cannot be allowed as each individual is considered distinct. It was due to this that the code makes use of an unconventional crossover using only a single parent, otherwise known as an asexual crossover.

A parent is first selected at random from a list of arrangements with an above average fitness, this will be referred to as $parent\ 1$. This parent arrangement is then split in half and stored in another variable that will be referred to as the $gene\ segment$.

$$gene\ segment = Parent\ 1\left(\frac{n}{2} : n\right)$$
$$where\ n = number\ of\ individuals$$

The $gene\ segment$ is then further shuffled and combined with the other half of $parent\ 1$ to form a child.

$$Parent\ 1 = [1,2,3,4,5]$$
$$Gene\ Segment = [3,4,5]$$
$$Gene\ Segment\ (Shuffled) = [4,5,3]$$

$$Child = Parent\ 1\ \left(\ 0 : \frac{n}{2}\right) + Gene\ Segment$$
$$= [1, 2] + [4, 5, 3]$$
$$= [1, 2, 4, 5, 3]$$

The child is then sent into the next generation, at which point it could possibly mutate and continue the algorithm further.

4) Mutation Function – The mutation function is a genetic operator that is used to maintain genetic variety in a population of genetic algorithm. With each iteration, all the combinations in the population will have the chance of being mutated. In the case of this particular mutation function, two seats within a combination will be swapped at random.

The probability for a combination within the generation to be mutated is set to 0.002 and is referred to as the mutation probability. The higher the mutation probability, the more likely it will be for a combination within the population to be mutated.

5) Termination Criteria – In the genetic algorithm, the program will keep iterating until it reaches the termination condition set in the program. For this case, there are two termination criteria that have been implemented.

Criteria one ends the run once the maximum number of generations has been hit. To further elucidate, the maximum number of iterations the program is set to go through is 1000. Once the loop counter reaches this maximum iteration, the program will terminate and print the optimal seating plan arrangement with the highest fitness score that was reached by the program.

Criteria two ends the run after there is no discernible change in the fitness after 300 iterations. The termination counter for criteria two iterates with the loop and only resets back to 0 if there is a change in value within the happiness list. This criterion is also referred to as convergence.

6) Elitism Selection – The selection of next-generation chromosomes within the algorithm is highly dependent on the elitism selection strategy. Individual combinations are effectively compared to one another with respect to fitness, with only the individual with max fitness being put it into the final list elitism winners. This elitism winners will eventually consist of all the best combinations from each generation, at which point a final elitism selection is called to print the combination with the maximum fitness.

7) Parameter Definition – During the initial runs, there are various parameters that require testing to give rise to an optimal value, namely, the population size, number of iterations, and convergence values.

In the case of population size, the value of 25 was chosen as there are 5! unique combinations for the seats and the number generates a large and consistent enough population to allow the algorithm to reach an optimal value without having to make use of an unnecessarily large number. It is to be noted that a larger population size would also result in a higher computational time.

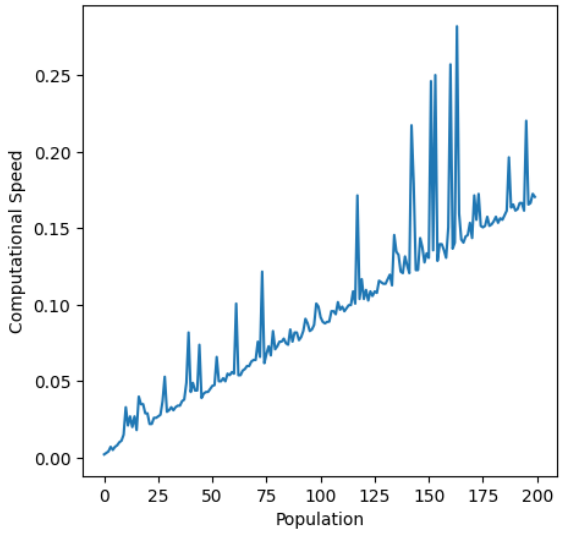

Figure 2.1 – Computation Speed vs Population Size

As seen in figure 2.1, due to the existence of convergence, the number of iterations is not constant throughout each run. Therefore, while the computational speed does notably increase with respect to the population size, the entropy of the genetic algorithm could possibly result in a higher or lesser than expected computational speed.

With regards to the number of iterations in this case, the value 1000 was chosen primarily since it provides a good benchmark for analysis. Furthermore, it also allows the convergence value to increase alongside it, as both values are to be proportional to one another. This is surmised as the convergence value is roughly 30% of the number of generations. In the case of 1000 iterations, the convergence value would in turn be 300. This would imply that the program is terminated once a single fitness value makes up 30% of the population over continuous iterations.

III. Result – Genetic Algorithm

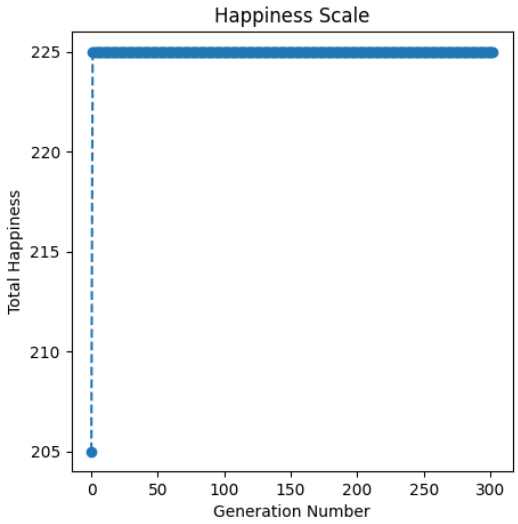

Figure 3.1 – Visualization of Happiness Scale

In figure 2.1, we can observe that the total happiness of each generation is plotted, and eventually hits an optimal value of 225. This value stays constant for 300 more iterations and eventually triggers termination based on convergence. The last iteration that is hit prior to termination being as follows.

```
Best of Generation  307
['B', 'C', 'D', 'A', 'E']  | Happiness:  225
```

Figure 3.2 – Output of Final Iteration

As seen above, figure 3.2 shows a sample output of the final iteration at 307th generation.

```
['B', 'C', 'D', 'A', 'E'] , with a happiness score of:  225
```

Figure 3.3 – Optimized Seating Arrangement

Figure 3.3 shows the final output of the program that follows suit and prints the combination with the maximum fitness value. While the output given does give a perfectly optimized seating arrangement, it should be noted that there are various combinations of seats that could result in a maximum fitness score of 225.

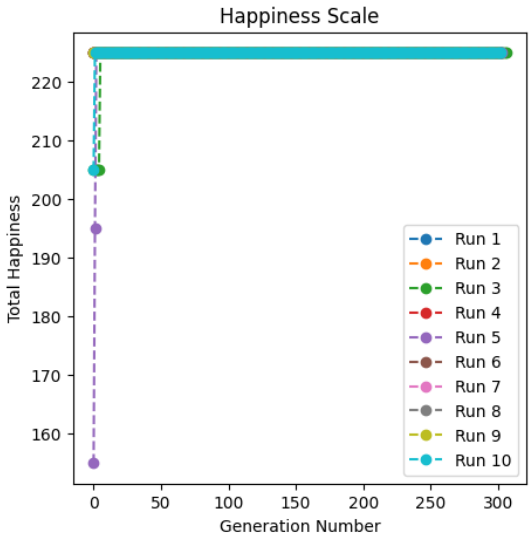

Figure 3.4 – Comparison of Multiple Runs

By analyzing figure 3.4, one can infer that over the course of 10 trial runs, the final optimized value reached was unanimously 225. The only discrepancy lies in the beginning of each iteration, at which, some have differing happiness values for their starting generation.

It is through these multiple trial runs that one can conclude the maximum happiness value is 225 and that the optimal seating arrangement can be represented by multiple seating arrangements, i.e. [A,D,C,B,E], [E,A,D,C,B], and more.

Thus, resulting in the most optimized seating to be, but not limited to, [B, C, D, A, E] with a total happiness value of 225.

IV. Analysis & Discussion – Genetic Algorithm

Referring to the figures in section III. Result – Genetic Algorithm one can conclude that using the initialized parameters, the final optimized result is consistent provides a perfectly optimized seating arrangement. As there is an added level of entropy when making use of genetic algorithms as well as the existence of more than one optimal seating arrangement, the end result may vary ever so slightly with each run.

Furthermore, this entropy also gives rise to the possibility of the algorithm coming across the optimal seating arrangement within the first few generations.

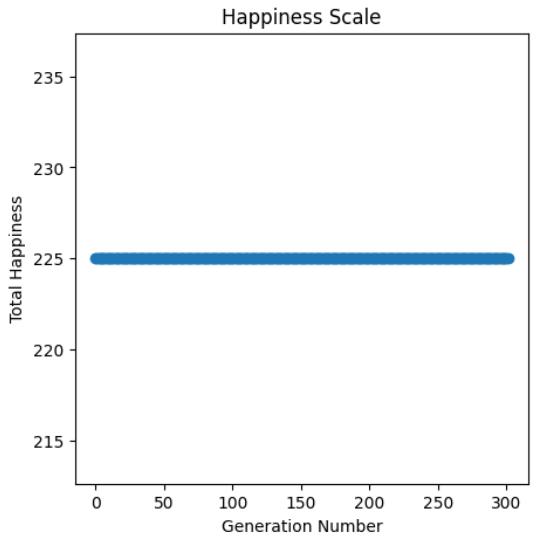

Figure 4.1 – Instant Optimal Seating

As can be seen in figure 4.1, coming across an optimal seating within the early generations will result in the algorithm terminating very quickly as there is little to no variation in maximum happiness for the future generations. As such, the resulting graph will be a straight line across the maximum happiness value 225.

V. Particle Swarm Optimization

A. Algorithm explanation

Particle swarm optimization (PSO) is a bio-inspired computational-intelligence technique to handle computationally hard problems based on the emergent behavior of swarms [3]. Swarm intelligence is the collective behavior of decentralized, self-organized systems. Such systems comprise a population of cooperating particles. Although there is no supervising control, this leads to 'intelligent' global behavior that is hidden to the individual particles.

PSO starts as a swarm of particles, where each particle is initially placed randomly in a particular position in the search space. Each position in the search space represents a potential solution. The initial velocities, and personal best positions can either be specified or more commonly, set to 0. In the best case for initialization, the positions of particles are initialized to uniformly cover the search space. It is important to note that the efficiency of the PSO is influenced by the initial diversity of the swarm. If certain regions of the search space are not covered by the initial swarm, the PSO will have difficulty in

finding the optimum if it is located within an uncovered region.

After initialization, the PSO algorithm will calculate the fitness of each particle. The fitness is calculated to compare the particles and identify the best position. The fitness function is calculation which characterizes the optimization problem and is how algorithm can determine the quality of a position. A function evaluation (FE) refers to a calculation of fitness function. The fitness evaluation is performed on all particles. The local best position of each particle and the global best position is also compared and overwritten in this process.

After evaluating the fitness of all particles and setting the local best position and global best position, the particles will begin to make their move. To do so, each particle must first calculate their new velocities. The velocity has two components. The first component is a cognitive component towards the particle's own current personal best. The second component is a social component towards the current best of the particle's entire social network. Particles "remember" their personal best location and the global best location in their social network. Each component is weighted by a random number in the range $[0, \phi_i]$, where the acceleration coefficients ϕ_i balance exploration versus exploitation in the optimization. The new velocity $v_i(t+1)$ is equal to the current velocity plus the influence of personal best and global best. To reproduce a particle to the next generation, the particle must move to a new position. To do so, its new velocity $v_i(t+1)$ is added to its current position $x_i(t)$.

Across multiple generations, the particles "fly" around in the search space according to its own current position and velocity. The direction towards which each particle "flies" is based on its own experience and the experience of the neighbors. Eventually, convergence is achieved through progressive particle flight contraction by mechanisms such as velocity clamping, inertia weight or constriction. Altogether, this is to reduce the possibility of particles flying out of the search space or exploring too much and not converging.

After all particles has made their move, the PSO algorithm will reiterate by looping back to the fitness evaluation process and set personal and neighborhood best positions accordingly. After that, the algorithm will update velocity and position of all particles yet again. This process or looping is repeated until a termination condition is triggered. Here are some examples of the termination criteria that can be considered:

- Terminate when a maximum number of iterations has been exceeded
- Terminate when an acceptable solution has been found
- Terminate when no improvement is observed over a few iterations
- Terminate when the particles are too close together
- Terminate when the change in fitness is approximately zero

When selecting a termination criterion, it should not cause the PSO algorithm to prematurely terminate, since particles have yet to converge, and suboptimal solutions will be obtained. Next, the termination criteria should protect against oversampling of the fitness. If it requires frequent calculation of the fitness function, computational complexity of the search process can be significantly increased.

B. Algorithm Variant Chosen

In this assignment, global best PSO, also known as gbest PSO, was implemented with inertia weight. In gbest PSO, the neighborhood for a particle is the entire population and all particles affect each other directly. With global best particle swarm optimization, the position update function is given by:

$$x_i(t-1) = x_i(t) + v_i(t-1)$$

Where:

- $x_i(t-1)$ = particle's new position
- $x_i(t)$ = particle's current position
- $v_i(t-1)$ = particle's velocity

And the velocity function is calculated by:

$$v_i(t+1) = wv_i(t) + \alpha_1\beta_1(t)\big(p_i(t) - x_i(t)\big) + \alpha_2\beta_2(t)\left(p_g(t) - x_i(t)\right)$$

Where:

- $v_i(t-1)$ = particle's new velocity
- w = inertia weight
- $v_i(t)$ = particle's current velocity
- x_i = particle's current position
- p_i = previous best position of individual
- p_g = previous best position of group

- α_1, α_2 = positive acceleration constants for personal best and group best
- β_1, β_2 = random values between 0 and 1

C. Implementation

1) Search space – The search space is the range of position values that the particles can explore around to find the optimal position/value. In this case, the particles are finding the best time a person should move according to costs and renovation level. The range of variables and value ranges given are:

- *days* - ranging from 1 (Monday) to 7 (Sunday)
- *hours* - ranging from 0 to 23
- *minutes* - ranging from 0 to 59

Since the variables are a function of time, instead of holding three different position values with three different range values, the time values can all be simplified into minutes ranging from 0 to 10079 (0 represents Monday 00:00 and 10079 represents Sunday 11:59). When needed, the *day*, *hour* and *minute* values can simply be calculated.

2) Fitness function – The fitness function is a performance measure to test how good the particle's position value is. As mentioned previously, the optimal solution would be a time where the cost for moving, and accommodation is as low as possible whilst renovation level is as high as possible. The formula provided for each variable is as such:

1. Renovation level:

$$p = \frac{T^2}{126} + \frac{T}{63} + 0.5$$

Where:

- p = renovation level
- T = current *day* including decimals

Since the position value's convention was modified, the formula for T provided by the problem is no longer applicable and thus was modified and simplified to:

$$T = x/d_m$$

Where:

- T = moving day, including decimals
- x = particle's position value (time in minutes)
- d_m = constant value of number of minutes in a day, 1440

2. Accommodation cost:

To calculate the cost for accommodation, no formula was provided but two statements stating, "For any part of a day you stay in the current accommodation, you will need to pay the rental rate of RM 30 per day." and "For each day staying in the new accommodation, you need to pay RM 25." A formula was formulated from these statements as such:

$$a = \big(30 + (T_2 * 30)\big) + ((7 - T_2) * 25)$$

Where:

- a = accommodation cost
- T_2 = moving day, excluding decimals

Formula to calculate T_2 is as such:

$$T_2 = x//d_m$$

Where:

- T_2 = moving day, excluding decimals
- x = particle's position value (time in minutes)
- d_m = constant value of number of minutes in a day, 1440

3. Moving cost:

$$\emptyset = 50\cos(\frac{12\pi t}{24}) + 50\cos(\frac{8\pi t}{24}) + 150$$

Where:

- $\emptyset$ = moving cost
- t = moving time (in hours including decimals)

Much like the previous case, since the position value's convention was modified, the formula for *t* provided by the problem is no longer applicable and thus was modified and simplified to:

$$t = (x\%d_m)/h_m$$

Where:

- t = moving time (in hours including decimals)
- x = particle's position value (time in minutes)

- d_m = constant value of number of minutes in a day, 1440
- h_m = constant value of number of minutes in an hour, 60

At first glance, the fitness function can be just as simple as taking the renovation level and subtracting the combined cost of accommodation and moving as such:

$$F = p - (a + \emptyset)$$

Where:

- F = fitness value
- p = renovation level
- a = accommodation cost
- $\emptyset$ = moving cost

However, further investigation leads to a discovery that the renovation function provided only computes values ranging from 0.5 to 1 whilst the cost for moving ranges from 68.29 to 50 and cost for accommodation range from 205 to 235. If this simple fitness function was used, the priority for renovation level will be significantly outweighed by the priority for lowest cost as its value is small relative to the costs. To fix this issue, a new fitness function needs to be formulating.

Instead of creating an entirely new formula, a modification to the previous formula was made. The solution was to scale the values to the same range so that both the cost and renovation level gets equal priority. To do this, either the renovation level can be scaled up to the range of cost, or the sum of both costs can be scaled down to the range of renovation level. The latter option was chosen for the implementation. The new fitness function will be as such:

$$f = p + (1 - \frac{(a + \emptyset)}{((a_{max} - a_{min}) + (\emptyset_{max} - \emptyset_{min}))})$$

Where:

- f = fitness value
- p = renovation level
- a = accommodation cost
- a_{max}, a_{min} = constant value of the maximum and minimum cost of accommodation, 235 and 205, respectively.
- $\emptyset$ = moving cost
- $\emptyset_{max}, \emptyset_{min}$ = constant value of the maximum and minimum cost of moving, 250 and 68.29, respectively.

3) Termination conditions – In gbest PSO, the program will keep iterating until it reaches one of the termination conditions. Two of the most termination conditions were implemented in this program.

The first termination condition is the maximum iteration condition. A maximum allowed iteration is set before the algorithm begins. If the algorithm reaches the maximum allocated iteration, the program will terminate the algorithm even if it has not completed searching.

The second termination condition is a distance measure. Distance of particles can be measured from different formulas and specified limit must be set. If the calculated particles' distance is lower than a specified limit, it means that the particles have converged onto a point and has found a solution. For this program, the average difference of fitness and distance were used. The formula used for both calculations are the same as is written as such:

$$d_{avg} = \frac{\sum_{k=1}^{n_k} x_k - x_{avg}}{n}$$

Where:

- d_{avg} = average difference
- x_k = fitness/position of particle
- x_{avg} = average fitness/position of all particles
- n = number of particles

Other termination conditions were not implemented due to the following reasons:

- Terminate when acceptable solution has been found – As this is an optimization problem, an acceptable solution is not known beforehand and thus, making this condition impossible to implement.
- Terminate when there are no improvements or nearly zero change in fitness – Particles are constantly moved every iteration and thus, fitness will always change. If there is no change in position, there will be no change in fitness. If particles do not change positions, it means that the particles have converged on an optimal location, and the algorithm will be terminated by the second termination condition.

4) Algorithm modification – There were two modifications made to help improve the performance of the algorithm.

The first modification was to the process of initialization. By default, the process of initializing the particles is just to randomly place the particles into the search space. This might cause a scenario where the placement of the population is skewed to one side, causing the other side to be unreachable and reduces exploration. To eliminate the possibility of this scenario from occurring, a new initialization process is derived. In the new initialization process, the population size and search space are split into equal segments. At each segmented search space, the segmented population is randomly spawned in. To verify if the segmented initialization is better than complete random initialization, a test was conducted with the different initialization process. Each initialization process was tasked to generate 50 particles in a search space of values ranging from 0 to 1,000,000. The results of the test can be seen in the figure 5.1 below.

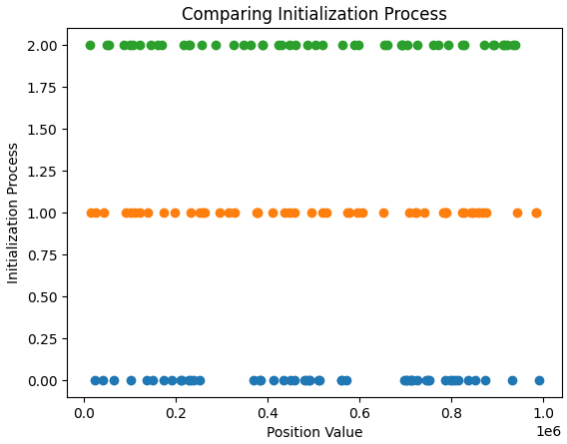

Figure 5.1 – Positions Generated from Different Initialization Process

Dots on figure 5.1 represents the position value of a particle. The blue dots were generated using the complete random initialization process while, the orange and green dots were generated with the new initialization process with splits of 5 and 10 segments respectively. From the results generated, it can be concluded that as the number of segments increases, the particles were distributed more evenly, and any gaps created by the randomness are smaller. Thus, proving that the new segmented initialization is better than the complete random initialization.

The second modification made was to alter the exploration and exploitation parameters overtime. When the movement of particles were visualized, a noticeable pattern became apparent. Even after the optimal value is found, some particles were still struggling to converge at the global best point. This is due to some particle's local best influence pulling it away from the global best point. These particles with equally strong attraction towards both the global best and local best points will begin hovering back and forth until a slingshot effect occurs and shoots the particle to either one of the points. If the slingshot effect shoots the particle towards the local best point, the particle will simply return to its hovering position and hovers until it is shot towards the global best point. This causes a serious problem as the particles are taking up iterations and making no progress or new discoveries. To resolve this issue, inertia weight was implemented into the program. Inertia weight was introduced into the algorithm to control and balance the exploration and exploitation. Higher inertia weight value means better exploration but lower exploitation, and vice versa. It is common for inertia weight to decrease over time to help converge the particles. To do so, the iterations are split into segments and the values of inertia weight is reduced when entering a new segment. Although implementing inertia weight did help with reducing the amount of hovering occurrence, the presence of hovering particles is still significant. Thus, instead of only reducing the inertia weight over iterations, α_1 is also decreased to further decrease hovering and α_2 is increased to force the particle to converge towards the global best point. As a result of this modification, the amount of hovering, wasted iteration, and computational time of the program is significantly reduced.

5) Parameter definitions – During implementation, some parameters require some manual testing and configuration. Examples of such parameters are number of particles, acceleration constants, inertia weights, position limits, and other parameters for the termination conditions.

Firstly, the number of particles needs to be set such that there are enough particles to be placed around the search space to explore most of its values. This means that the number of particles is directly proportional to exploration. However, larger number of particles also comes with a heavier computational load. Therefore, there is a need to consider better exploration or better computational speed. To test whether an optimal value for population size could be discovered, a test was conducted to compute the computational time of the program as population size

increments by 10 starting from 10 to 1,000. To minimize the effect of randomness, the beta values were fixed to 1 and the *random* module is reseeded every time the population number is altered. The result of the test is shown in figure 5.2 below.

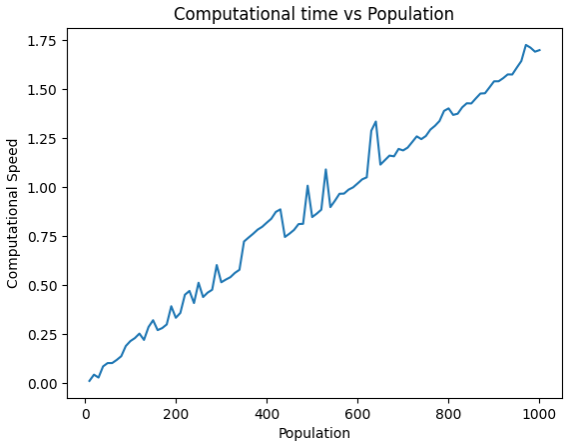

Figure 5.2 – Computational Time vs Population Size

As shown from figure 5.2, computational time increases almost linearly with population size thus, there is no conclusive method to determine optimal population size. As a result, a random number was simply chosen for the population size. The random number chosen for this project was 100 as it is not too small to restrict exploration and not too large to cause high computational time.

Secondly, the alpha values need to be set such that the exploration and exploitation of the algorithm is optimal. $\alpha_1$and α_2 are the positive acceleration constants that influence the particle's cognitive and social components respectively. Higher α_1 value provides better exploration while higher α_2 value provides better exploitation. As there is no decisive method to discover the optimal values of alpha, a decision to provide equal priority for both exploration and exploitation was made. Therefore, both alpha values are initially set to 0.5. As mentioned previously, these values will be changed over iterations. When the iteration for change is reached, the value of α_1 will be reduced by 0.1 and the value of α_2 will be increased by 0.1. This provides more exploitation and reduces exploration and thus, forces the particles to converge to the global best point, preventing hovering.

Thirdly, inertia weight needs to be added and scaled as iterations increases. As mentioned previously, higher inertia weight value means better exploration but lower exploitation. Therefore, as to allow full exploration without constrictions, the starting inertia weight of the algorithm was set to 1. When the iteration for change is reached, inertia weight will be reduced by 0.2.

Fourthly, is the number of maximum iterations allowed for the program to run. Since many efforts of modifying and optimizing the algorithm has been made, the algorithm does not need much iteration to find the optimal value for the problem. Rather, most of the iterations will be used to converge all the particles together to the optimal point. As such, the maximum iteration can be set as low as 80 iterations. But to offer some form of extra safeguard, the maximum number of iterations is increased to 100 iterations.

Finally, the maximum values of average difference allowed for termination must be determined. The values must be set such that when termination condition is triggered, most of the particles have converged onto the same point with some acceptable margin. Lower maximum average difference value forces the particles to be more concentrated around a point before the algorithm can be terminated. Thus, lower maximum average difference allows for more accurate particle convergence point. However, if the value is set too low, it will require all particle to be exactly on the same point before the algorithm can terminate. This can cause the algorithm to go on for a long time as the particles will move slower as iterations increases. As such the maximum average difference values allowed are set to 0.01.

VI. Result – Particle Swarm Optimization

After all configurations and optimization of the global best PSO algorithm has been completed, it is finally time to test the program. A program was made to run the algorithm 100 times and generate figures from the figures. The results generated from the program is as follows.

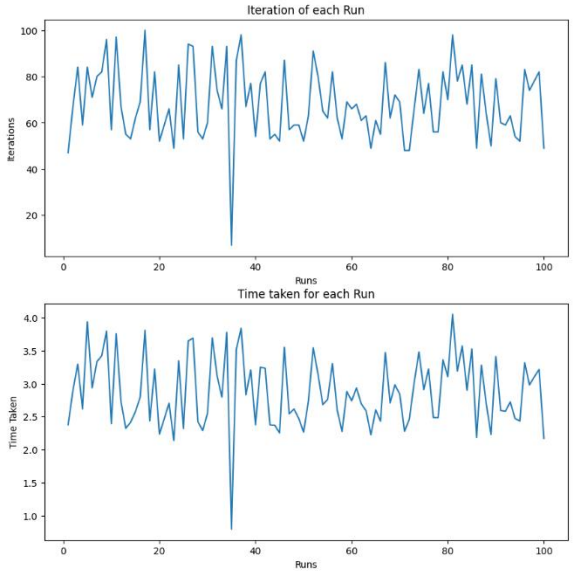

Figure 6.1 – Iteration and Time of Each Run

Figure 6.1 shows the time taken, and iteration used for each run. A few observations can be made from the figure. Firstly, both the iteration and time taken figures produce the same line. Secondly, most runs take at least 40 iteration and 2 seconds to complete its run except for run 35, where an abnormality occurred. Finally, no pattern from the line can be observed and thus, both iteration and time taken are still random and may be caused by some factors in the algorithm.

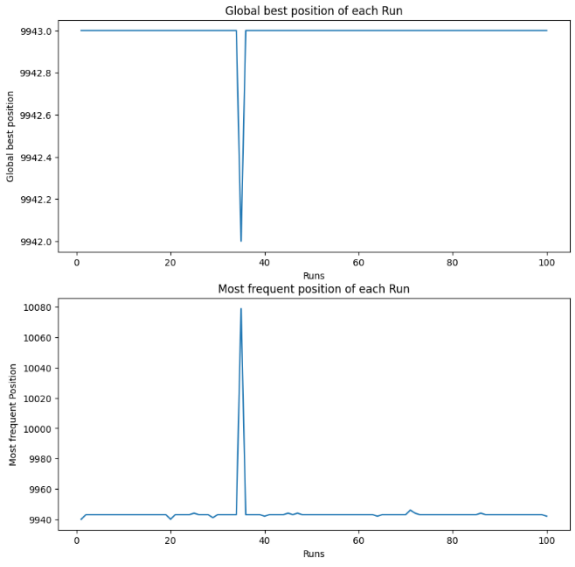

Figure 6.2 – Global Best Position and Most Frequent Position of Each Run

Figure 6.2 shows the global best position and the convergence point (the point with the most particles on it) of each run. A few observations can be made. Firstly, a major dip at run 35 leads to further confirmation that an abnormality has occurred. Secondly, at every run besides run 25, the global best position found seems to always be at point 9943, indicating that the algorithm is producing a stable result. Finally, from the most frequent position figure, it can be observed that most of the particles either converge directly onto or very close to the global best point.

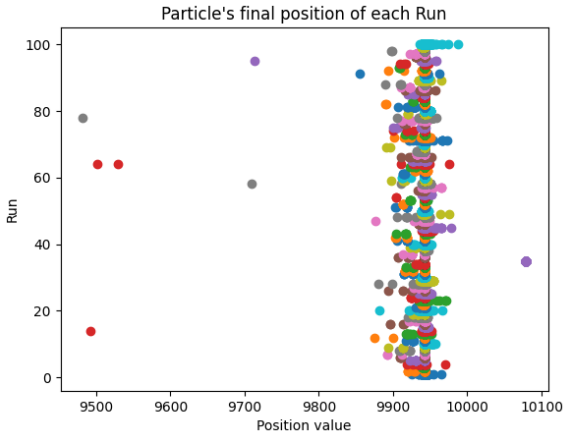

Figure 6.3 – Final Position of Every Particles of Each Run

Figure 6.3 shows the position of all particles in the final iteration of each run. A few observations can be made from the figure. Firstly, on the abnormal run, run 35, all particles have converged onto the edge of the search space, point 10079, indicating that the particles have all flown out of the search space and was pulled back. Secondly, all particles are tightly packed together, showing that the particles have converged on a solution. Finally, every run has a similar convergence point, around point 9943, showing that the convergence point is the final optimal solution.

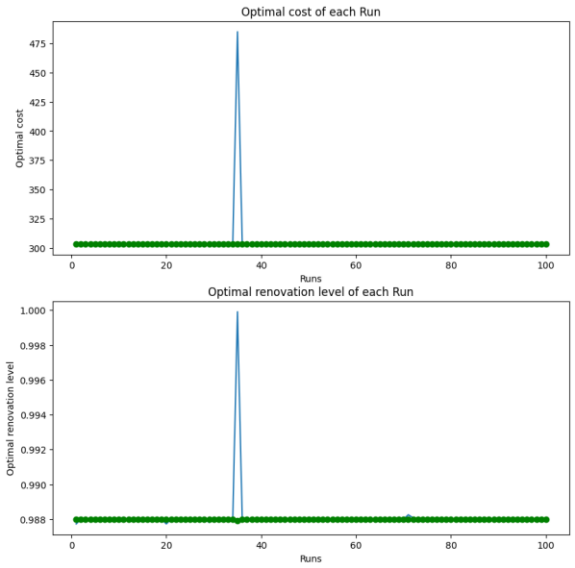

Figure 6.4 – Optimal Cost and Renovation Level of Each Run

Figure 6.4 shows the total cost and renovation level at the optimal time that the algorithm has found. The green dotted line represents global best position while the blue line represents the convergence point. Although the convergence point differs slightly from the global best point (as seen from figure 6.2) for some runs (excluding the abnormal run, run 35), both the total cost and renovation level is almost the same for both the convergence point and global best point.

Thus, the optimal time to move out can be said to be around 9943 minutes from Monday 00:00, which is equivalent to Sunday 21:43. At this time, the combined cost for accommodation and moving will be RM 303.30 and the renovation level of the new location will be 98.8% completed.

VII. Analysis & Discussion – Particle Swarm Optimization

From the figures shown in section *VI. Results – Particle Swarm Optimization*, the algorithm have been producing consistent results. The optimization modifications have also helped to improve the algorithm significantly to reduce total iterations required for each run and balance exploration and exploitation factors. However, there are still some randomness to the algorithm that causes inconsistent number of iterations required. This randomness can be justified to be caused by the beta values and the particle's initial position.

A flaw still exists in the program that may still cause some abnormal runs like run 35. As exploration is at its peak in the early stages of the algorithm, particles are flying around the search space with high velocities. There is a rare case that the particles will all fly out of the search space and be pulled back into the edge of the search space. This causes an issue as all particles will be set onto the same point at the edge and trigger the second termination condition. In turn, this causes the algorithm to end prematurely, and thus, the particles fail to search for the optimal location. A solution to bypass this issue is to pull the particle back into a random location in the search space rather than the edge but this also causes another issue. If the optimal location of the particle is near the edge of the search space, the particles cannot converge onto that point as the particles will constantly fly out of the search space and be pulled back into a random location in the search space.

VIII. Conclusion

The paper described the approach to two problems using a genetic algorithm and the particle swarm optimization algorithm which are widely applied in the field of Computational Intelligence. The genetic algorithm was used to determine the optimal seating position in a group of individuals whilst particle swarm optimization was used to determine the optimal time for a person to move from their accommodation.

Through the employment of a genetic algorithm to study the happiness levels of various individuals within a group in respect to one another, the most optimal seating arrangements were generated. Whilst carrying out iterations of the algorithm, we were able to find that there was more than one arrangement that would satisfy all parties, thus allowing more choice if more restrictions were imposed on individual preferences when being seated. However, caution must be exercised when setting the parameters prior to testing as this could lead to possible discrepancies and flaws in the seating arrangements. With the testing done, we can observe the possible errors that may occur and how we can overcome them to be fully optimized in the future.

By having produced consistent results, the Particle Swarm Optimization algorithm enabled us to determine the optimal time for a person to leave the accommodation and move in. We were also able to learn of the inconsistencies and randomness that occurs with the algorithm from the results as well as

flaws that can occur in runs that may be detrimental to the result. However, in spite this, testing had been successful, and we were still able to produce results that were favorable.

REFERENCES

[1] V. Mallawaarachchi, "Introduction to Genetic Algorithms — Including Example Code," *Medium*, Mar. 01, 2020. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (accessed Oct. 01, 2021).

[2] "The Basics of Genetic Algorithms in Machine Learning," *Engineering Education (EngEd) Program | Section*. https://www.section.io/engineering-education/the-basics-of-genetic-algorithms-in-ml/ (accessed Oct. 01, 2021).

[3] Kennedy, J. & Eberhart, R. Particle swarm optimization. In *Proc. ICNN'95—International Conference on Neural Networks* Vol. 4, 1942–1948