AEROSP – 588
Final Project

Prit Chovatiya

Drone Delivery Schedule Optimization
An Investigation

Introduction

Background and Motivation

The e-commerce industry is are predicted to grow a staggering $4 trillion by 2023 [1]. Consequently, online retailers, in order to compete in this arena, attempt to deliver the best possible products in the least amount of time. Last-mile delivery optimization plays a crucial role in achieving this feat and companies like Amazon.com, Inc. are actively seeking to use fleets of drones to deliver products at the customer's doorstep. Scheduling the delivery in most efficient manner with the least amount of wait time for the customer is key element to this. Even though the "travelling salesman-like" problem has been already tackled by companies in the past, the usage of drones and certain limitations with them presents a new challenge with a different set of constraints. Particularly, a drone's batteries have to be charged, forcing them to return to the warehouse repeatedly in a given day while also meeting delivery-time constraints.

As freight companies expand to new cities, the task of finding an optimal delivery schedule with drones is particularly challenging. Specifically, the scale of the problem grows exponentially with the increase in the number of locations (shown in Figure 1). For instance, making a delivery schedule for 1 warehouse, 8 delivery locations and a possibility of charging after each delivery has $8! \times 2_7 = 5,160,960$ potential delivery routes!

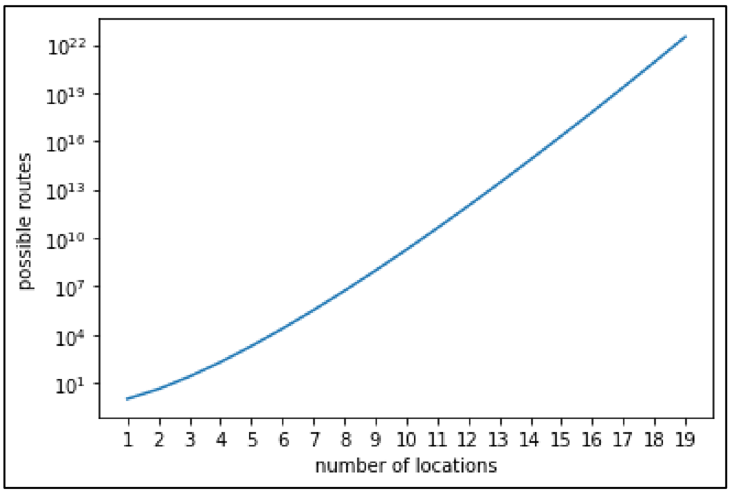

Figure 1: Possible Routes with Locations

Just adding two new locations for the same problem, the delivery routes grow to a whopping 1,875,645,600 possibilities! Given the sheer scale of this problem, it is rather unrealistic to test every permutation of the routes calling for a faster method. This problem is certainly an appropriate candidate for an optimization-based investigation that can evaluate and select better designs automatically. Hence, a detailed study on the same is presented hereby.

Problem Formulation

Optimization Problem Statement

Consequently, the problem for an optimization strategy is formulated in a manner that is easy to implement, robust and scalable. The problem can be summarized via this question: *For a given set of packages to be delivered to different locations, what is the most optimal departure schedule consisting of order of deliveries and charging times for the drone?* The problem stated above is formulated so that the objective function, essentially a cost function which is being minimized. The cost function is a linear combination of two terms- 1) the total distance travelled by the drone and 2) number of charging stops used in a trip. This function can also be modified to conduct a multi-objective optimization for a pareto front. The formulated problem is given below:

$$minimize \rightarrow f(x)$$

$$\begin{aligned} with\ respect\ to: \quad & x_i\ , y_j \quad i = 1, 2, \dots n \\ & \qquad\quad\ j = 1, 2, \dots n-1 \\ subject\ to: \quad & sb_{i\ to\ i+1} \geq b_{required} \\ & x_{i+1} \neq x_i\ for\ all\ i \\ variable\ bounds: \quad & 1 \leq x_i \leq n \\ & 0 \leq y_j \leq 1 \end{aligned}$$

$Here, the\ cost\ function\ f(x)\ is\ given\ by: -$

$$f(x) = \alpha \cdot \sum_{i=1}^{n} \left[\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \right] + \beta \cdot [\#\ of\ charging\ stops]$$

$where:$

$$\begin{gathered} x_i \rightarrow order\ of\ deliveries \\ y_j \rightarrow charging\ decisions\ after\ each\ delivery \\ n \rightarrow number\ of\ locations \\ b \rightarrow battery\ level;\ sb \rightarrow stored\ battery \\ \alpha, \beta \rightarrow scaling\ factors \\ (x_i, y_i) \rightarrow current\ location\ coordinates \\ (x_{i-1}, y_{i-1}) \rightarrow previous\ location\ coordinates \end{gathered}$$

As seen from the problem formulation above the design variables for this problem will be sequential in nature. For instance, in a case of 8 locations, the two-design variables will be in the following vector form:

$$x_i = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$$

$$y_j = [y_1, y_2, y_3, y_4, y_5, y_6, y_7]$$

Here, x_1 represents the first location to be visited, followed by x_2 and so on. Secondarily, y_j represents a decision "to go charge at warehouse" versus "continue without charging" by parameters '1' and '0' respectively. An example of how these design variables are interpreted is shown below.

For this particular investigation, 8 locations and a warehouse are simulated, which is plotted in Figure 1. The coordinates of these locations and the warehouse are shown in Table 1 and the locations themselves are visually represented in Figure 2.

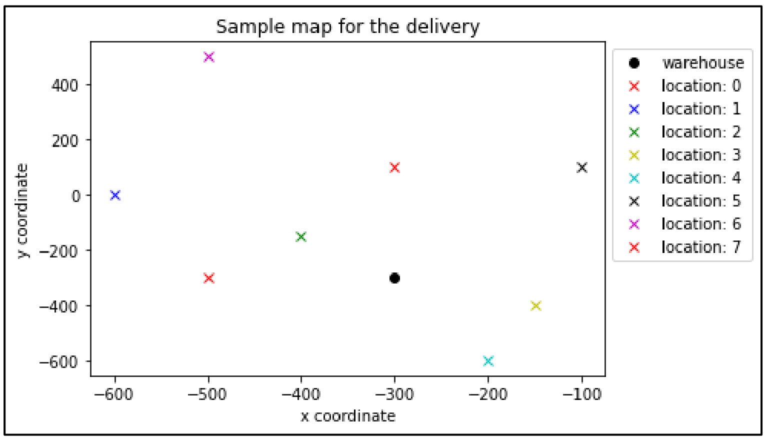

Figure 2: Sample map for an 8-location delivery problem

Location	Coordinates
# 0	[-300, 100]
# 1	[-600, 0]
# 2	[-400, -150]
# 3	[-150, -400]
# 4	[-200, -600]
# 5	[-100, -100]
# 6	[-500, 500]
# 7	[-500, -300]
Warehouse	[-300, -300]

Table 1: Location Coordinates

Considering an example, in a case of 8 locations, for the two-design variables:

$$x_i = [7, 3, 6, 2, 4, 0, 5, 1]$$

$$y_j = [1, 0, 0, 1, 0, 0, 0]$$

Here, it should be noted that according to y_j design variable, two charging stops are recommended, the first one after visiting location 7 and the second one after visiting location 2. Furthermore, all the charging stops occur at the warehouse and consequently the drone flies back mid-trip. All the trips start and end at the warehouse representing the real-world scenario. For indexing purposes, location numbers range from 0 to 7 instead. The simulated trip can be shown by the following:

```
Going from warehouse: (-300, -300) to starting location # 7 at: (-500, -300)
--------------
Warehouse stop between loc # 7 at: (-500, -300) to loc # 3 at: (-150, -400)
--------------
Directly going from loc # 3 at: (-150, -400) to loc # 6 at: (-500, 500)
--------------
Directly going from loc # 6 at: (-500, 500) to loc # 2 at: (-400, -150)
--------------
Warehouse stop between loc # 2 at: (-400, -150) to loc # 4 at: (-200, -600)
--------------
Directly going from loc # 4 at: (-200, -600) to loc # 0 at: (-300, 100)
--------------
Directly going from loc # 0 at: (-300, 100) to loc # 5 at: (-100, -100)
--------------
Directly going from loc # 5 at: (-100, -100) to loc # 1 at: (-600, 0)
--------------
Going from last loc # 1 at: (-600, 0) back to warehouse.
--------------
Charged 2 times during the full trip.
Total distance travelled during full trip: 4624.206124985304
```

Figure 3: Sample input interpretation of an 8-location model

Another interesting consideration here, is that the design variable y_j also directly affects the objective function as minimizing charging stops saves time and improves efficiency. The function essentially returns a weighted objective value including the distance and the charging.

Problem Classification

The above problem can be classified into a multi-objective optimization – which includes minimizing the distance travelled by the drone and minimizing the amount of charging for a given trip. The design variables are discrete here as location indices and charging decisions have to strictly be integers for simplicity. There are two key constraints imposed on the design: 1. the battery stored in the drone always has to have the potential of returning back to the warehouse from any given point in the trip and 2. no location can be revisited after they have been visited once except the warehouse. A key assumption here is that the drone has the capability to carry payload for 4 deliveries, implying that it would be able to deliver 4 packages simultaneously during a trip without going back to the warehouse. With regards to the objective and the constraint function characteristics, the objective function is non-smooth. In terms of the linearity no certain statement can be made. The objective is certainly non-convex, multimodal and deterministic.

Models and Coupling

The model used in this problem to compute the objective function consists of multiple components. Two important objectives of minimizing the distance travelled and minimizing the amount of charging stops in a trip are indeed coupled. In a particular case, when the optimization algorithm is minimizing the distance travelled during a trip, the drone might have to make multiple charging stops at the warehouse, to make the trip possible. This implies that even though the distance is being minimized, there are still a certain number of charging trips required, which in turn increase the amount of distance travelled. Contrarily, when the optimization is minimizing the number of charging trips, the solution still has to feasible enough to visit all locations without failing to return to the warehouse with sufficient charge. In both the cases, the objectives are competing and need to be handled appropriately.

Optimization Algorithm

It was evident that due to the discrete nature of the inputs, unavailability of gradients and the discontinuity of the objective function, a gradient-free optimization was to be sought. Even though gradient free methods perform inefficiently on high-dimension problems, this method works well with less than 30 variables, which in our case is only 15. Historically, problems consisting of sequential optimization also have been dealt with evolutionary methods such as genetic algorithm due to their flexibility in representing discretized design variables and exploring a “wide” design space. Furthermore, since a solution for the aforementioned problem requires a global search, genetic algorithm is chosen for this problem. Even though alternative methods such as Nelder-Mead and DIRECT can be used, these methods would often have extraneous amounts of constraints to ensure the discrete nature of design variables, rendering it more complicated and tedious than the former. Hence, due to the simplicity and robustness of the genetic algorithm to function with discrete design variables, it was chosen as the primary method.

This method uses heuristically determined optimality criterion and iteration procedure while exploiting directly evaluated objective functions. The genetic algorithm is ‘population based’ which uses randomly generated ‘individuals’ consisting of that population. This population is reproduced iteratively, while filtering out the ‘fittest’ individuals (designs) every time based on the principle of biological evolution and "survival of the fittest". It involves three important components, namely: 1. selection followed by 2. crossover (reproduction) and finally, 3. mutation (natural variation). The implementation of this method is covered in the following section.

Problem Setup

A major classification of the genetic algorithms is dependent on the manner of encoding the design variables. They can be either 'binary encoded' or 'real-encoded'. Binary encoding is used to represent number, whether an integer or a real number, in the form of bits: '0's and '1's. Whereas, real encoding simply translates the numbers to real values themselves. Encoding is done to replicate the chromosome type of representation of various designs which are merged and modified in various ways to introduce either 'crossover' or 'mutation' in the design variables.

In our case, a 'binary encoded' implementation is presented, as this is one of the most efficient ways to represent integers, a characteristic which certainly dominates our design space. In this method, each design variable is represented with 'm' number of bits. Each bit, as stated above, has a value of either '0' or '1'. Various permutations of these bits tend to represent various numbers. For a given design variable $x \in [x_{lower}, x_{upper}]$, this design space can be divided into $2^m - 1$ intervals. From which, the precision of this representation can be found via the given formula:

$$\Delta x = \frac{x_{upper} - x_{lower}}{2^m - 1}$$

For our problem, two design variables: the delivery sequence and the charging decisions have to use different types of representation since both the variable have different bounds. In the case of the delivery sequence, $x_i \in [0,\ 7]$, representing the indices of each of the 8 locations. Since we wish to only have a precision of $\Delta x = 1$ between these variables, the number of bits used to represent this is given by:

$$\Delta x = \frac{7-0}{2^m - 1} \quad \rightarrow \quad 2^m = \frac{7-0}{1} + 1$$

Resulting in a 3-bit representation of the 8 locations. Table 2 shows each of the design variables for locations represented by a 3-bit representation.

Location	3-bit representation
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Table 2: 3-bit representation of x_i

Contrarily, in the case of the charging decision variable $y_j \in [0,\ 1]$, each of these values could be simply represented by '0's or '1's themselves. Finally, these two design variables can be represented into a ‘design string’ of 31 binary digits. For instance, a design $[x_i\ ,\ y_j]$ given below:

$$[x_i\,; y_j] = [0,1,2,3,4,5,6,7\,;\,0,0,1,0,1,0,1]$$

can be represented as:

000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 | 0 | 0 | 1 | 0 | 1 | 0 | 1

Upon encoding the design variables in a binary string representation, all the tasks of the genetic algorithm namely: selection, crossover and mutation can be conducted by string manipulation but there is still a need of decoding these designs for the objective function evaluations. For decoding these representations, the formula given below is used to convert bits into integers:

$$x = x_{lower} + \sum_{i=0}^{m-1} b_i \,.\, 2^i . \Delta\, x$$

Generating Initial Population

The first step in the implementation of the genetic algorithm is to generate a randomized design space with ‘N’ number of individuals, all grouped in a generation. As a general heuristic rule, the number of individuals ‘N’ in the initial population is usually more than an order of magnitude larger than the number of design variables. To randomize the design generation for a binary representation, a random number in $r \in [0,\ 1]$ is selected. It is then probabilistically determined that if $r \leq 0.5$, then the bit will be chosen as a ‘0’ and if $r \geq 0.5$, then the bit will be assigned a value of ‘1’. This is done repeatedly, until a binary string of 31 bits is obtained representing one design in a population of N individuals. Overall, for an initial population of 1000 individuals, 31,000 bits have to randomly generated. Here, the population size N can vary and is heuristically determined which ensures maximum diversity in the design space.

Evaluating Fitness

Upon generating the initial design population, all the individuals are to be evaluated by the objective function to start selection process. To do this, the encoded initial population is decoded and fed into the objective function. Contrary to the other optimization methods, the genetic algorithm maximizes the objective. For our case, since the objective is being minimized, the objective function is converted to $F = \frac{1}{f(x)}$ resulting in the objective function given by:

$$F(x) = \frac{1}{\alpha \cdot \sum_{i=1}^{n} \left[\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\,\right] + \beta \cdot [\#\ of\ charging\ stops]}$$

Furthermore, to ensure that our designs are feasible, the objective function has to be transformed to a merit function which reflects the violation of constraints as well. In our problem, the two constraints:

$$c_1(x) \rightarrow\ sb_{i\ to\ i+1} \geq b_{required}$$

$$c_2(x) \rightarrow\ x_{i+1} \neq x_i\ for\ all\ i$$

These ensure that the drone has enough battery at all times to get to next location (c_1) and that the drone does not revisit any previously visited locations (c_2) respectively.

In the case of c_1, the drone starts the trip with stored battery $sb = 1500\, meters$. Here, the battery is linearly related to the distance, implying that a single charge can give the drone the ability to travel 1500 meters at once. The stored battery sb is repeatedly calculated at every step in the trip, yielding the following constraint violation values $c_{1,vio}$ if the drone has/doesn't have the sufficient battery at all the steps in a trip, given by the equations below:

$$c_{1,vio}(x) = 0 \qquad if\ sb_{i\ to\ i+1} \geq b_{required}$$

$$c_{1,vio}(x) = \left| sb_{i\ to\ i+1} - b_{required} \right| \qquad if\ sb_{i\ to\ i+1} \leq b_{required}$$

In the case of constraint c_2, a simple repetition check is used to compute the $c_{2,vio}$ values given by the following equations:

$$c_{2,vio}(x) = 0 \qquad if\ x_{i+1} \neq x_i\ for\ all\ i\ \ (no\ repetitions)$$

$$c_{2,vio}(x) = count[x_{i+1} = x_i] \qquad if\ x_{i+1} = x_i\ for\ all\ i\ \ (repetitions)$$

On the other hand, the two constraint violation values are combined into a penalty function given by the following equation:

$$C_{vio}(x,\, \mu_1,\, \mu_2) = \mu_1 . c_{1,vio}(x) + \mu_2 . c_{2,vio}(x)$$

Here, μ_1 and μ_2 are relative scaling constraints to ensure homogenous constraint violation penalties apply appropriately. Finally, a merit function $F(x,\, \mu_1,\, \mu_2)$ is used to include the constraint violation as a linear combination of the objective function and the penalty function given by :

$$F(x,\, \mu_1,\, \mu_2) = \frac{1}{\alpha \cdot \sum_{i=1}^{n} \left[\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \right] + \beta \cdot [\#\ of\ charging\ stops] + C_{vio}(x,\, \mu_1,\, \mu_2)}$$

This function is maximized and used to evaluate the 'fitness' for every individual in a given population. This method allows the optimization algorithm to conduct an unconstrained search on an implicitly constrained objective function making it simple to implement while ensuring feasibility of designs.

Selection

After evaluating fitness of all the design variables in the population, a selection procedure inspired from the concept of "survival of the fittest" is used to generate 'parent' populations. These parent populations are then used to generate 'off-springs' which populate the subsequent generation. This process improves the average fitness of a generation resulting in filtering of better designs. There are various methods for conducting selection such as tournament or roulette wheel selection. Hereby, the tournament selection method is used. In this method, two randomly selected individuals are evaluated and compared with each other, and the better design is chosen to be in the final parent pool. It results in a pool of N/2 parents from an initial generation consisting of N individuals. This is done twice to create two parent pools which are then used for crossover or 'mating' of parents from both the pools. This process is better shown in Figure 4 below.

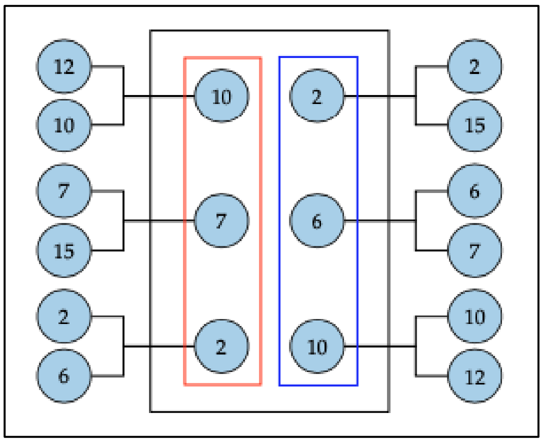

Figure 4: Tournament Selection

As it can be seen here, a tournament selection is conducted twice, with the two selection procedures resulting in the parent pools given in red and blue respectively.

Crossover
This process uses the two parent pools resulting from the tournament selection process and uses them to generate an 'offspring' population. In general, two individuals arise from a single parent pair resulting in a population of N individuals. This method can be completed using an operation called 'n-point crossover'. Here, the binary representation of one randomly selected parent from each parent pool is considered. For our problem, a 2-point crossover is done for the sequence design variable x_i. This process can be visualized using Figure 5 where parents from parent pool A and B are represented in red and blue respectively.

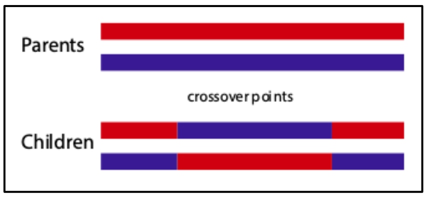

Figure 5: 2-point Crossover

Two offspring are generated by selecting the first and the third set of bits from parent red and selecting the second set of bits from parent blue and vice versa for the other offspring. In this way, characteristics from both parents are transferred to resulting offspring. The following example applies this strategy and results are shown below. It is to be noted that only 4 variables of 3-bit design representations out of 8 variables in x_i are shown below:

Parent A = | 000 | 001 | 010 | 011 | Offspring A = | 010 | 001 | 010 | 001 |
Parent B = | 111 | 101 | 110 | 100 | Offspring B = | 101 | 101 | 110 | 101 |

With regards to the charging decision design variable y_j, an alternate crossover strategy was used. This process alternately selected bits from both the parents similar to the 2-point crossover, for the binary representation of y_j shown below:

Parent A = | 0 | 0 | 1 | 0 | 1 | 0 | 1 | Offspring A = | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
Parent B = | 1 | 0 | 1 | 1 | 0 | 0 | 1 | Offspring B = | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

In this manner the offspring for the next generation were found.

Mutation

For the final operation of the genetic algorithm, mutation is an operation inspired by the occasional genetic mutation that occurs in a given population. It is essential for this method as it introduces an element of diversity in the population. Even though crossover and selection combine the characteristics of better designs into the offspring, mutation helps cover the gaps in some information which is lost in those operations. To make this possible, a 'bit-flip' strategy was used to introduce these mutations in a certain number of designs in a population. The example below shows this operation on a design:

Before mutation: 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 | 0 | 0 | 1 | 0 | 1 | 0 | 1
After mutation: 111 | 110 | 101 | 100 | 011 | 010 | 001 | 000 | 1 | 1 | 0 | 1 | 0 | 1 | 0

One of the most neglected but a rather crucial component of the mutation procedure is the mutation rate which impacts the number of individuals being mutated in every generation. There are many ways in which this hyperparameter is set which has different implications on the search. Extremely high mutation rates introduce abnormally high diversity into the population making this more of a randomized search. On the other hand, extremely low mutation rates can be responsible for leading the search into a local minimum. Furthermore, a constant mutation rate throughout all the generations can also produce inconsistent results. With regards to our implementation, a heuristically determined dynamic-mutation rate was adopted. This rate was decreased by some fixed percentage every 10 generations to ensure a robust search through the design space.

Overall Implementation

Upon defining the main operations for the genetic algorithm, a collective method of the algorithm is shown by the pseudo-code below. Firstly, the bounds for the design variables are inputted, which are used to generate the initial population. Each individual in the population is evaluated via the objective function and selection is conducted upon them resulting in two parent pools. These pools are inputted into the crossover function which outputs N offspring resulting in a new generation. Based on a specific mutation rate, some individuals in a population are then mutated. This process is repeated for a given number of generations counted by 'k' and the best designs are produced as the output.

```
Genetic algorithm
Input: Initial variables bounds
  $k = 0$
  Generate initial population $P_k = \{x^{(1)}, x^{(2)}, \dots, x^{(N_P)}\}$
  while Stopping criterion is not satisfied do
    Compute objective $f(x)$ for each $x \in P_k$              ▹ Evaluate fitness
    Choose points from population for the "mating pool"        ▹ Selection
    Create a new population ($P_{k+1}$) from the mating pool   ▹ Crossover
    Randomly mutate some points in the population              ▹ Mutation
    $k = k + 1$
  end while
Return: "Optimum" (best result found), $x^*$
```

Figure 6: Pseudo code for Genetic Algorithm

Optimization Results

Optimization of a Simplified Problem

To verify the correct implementation of the genetic algorithm, it is crucial to conduct 'sanity-checks' before the complete execution of the algorithm on a complex model. To enable this, the genetic algorithm is implemented on a simplified objective function given below:

$$f(x) = \alpha \cdot \sum_{i=1}^{n} \left[\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \right] + \beta \cdot [\# \, of \, charging \, stops]$$

It is to be noted that here, the penalty component for constraint violation has been omitted making this simply a minimization of the total distance travelled and the number of charging stops. For this simplified version, neglecting the constraints for battery storage and repetition of delivery locations, the algorithm should naturally converge to a design variable with the same repeated locations in x_i since this results in a minimal total distance. Furthermore, the number of charging stops are minimized by merely not having any charging stops at all. The result should be of the form, where x is the location which closest to the warehouse.

$$[x_i^* \, ; y_j^*] = [x, x, x, x, x, x, x, x \, ; \, 0, 0, 0, 0, 0, 0, 0]$$

Since, now we have an understanding of the nature of the expected result from the minimizer, we shall run the genetic algorithm and verify the same. Upon implementing this, the algorithm yielded an optimum design variable given below:

$$[x_i^* \, ; y_j^*] = [2, 2, 2, 2, 2, 2, 2, 2 \, ; \, 0, 0, 0, 0, 0, 0, 0]$$

This is indeed the same as the expected result. Here, the optimization yields, $x = 2$, since location # 2 is the closest to the warehouse resulting in the minimum total distance. Furthermore, the convergence plots for the algorithm are given below. Here, in Figure 7, the average fitness of every generation is plotted with the number of generations. It can be clearly seen that the algorithm, converges with a design with maximum fitness for the last few generations.

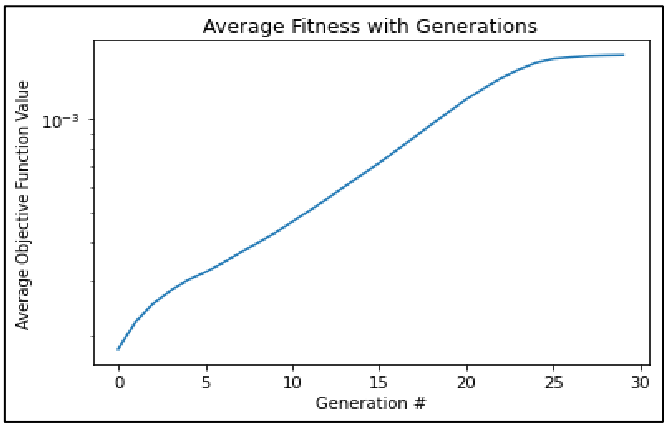

Figure 7: Average fitness with generations

Additionally, the distance of the best design from every generation has been plotted below in Figure 8. As it can be verified the algorithm converges on the minimum distance.

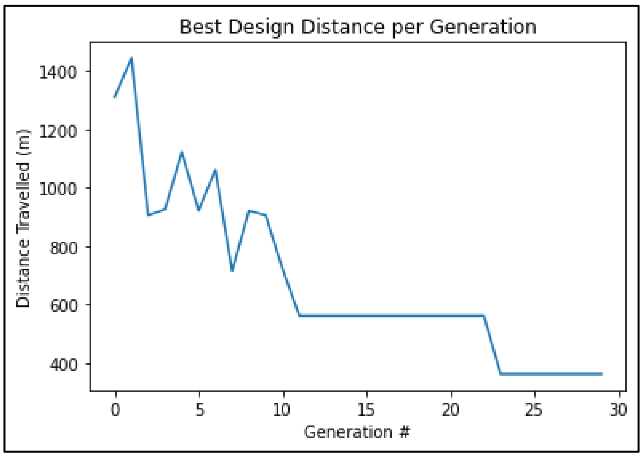

Figure 8: Average fitness with generations

Now, since the algorithm functions as expected for a simplified model, it can be used to optimize the complex model.

Algorithm Justification

As seen from the results of the optimization of the simplified problem, the genetic algorithm proves to find a solution quickly. A population size of 5000 individuals and 30 generations was used. This implies that the algorithm took only 150,000 function evaluations to find the minima for a problem which consists of more than 5,160,960 possible designs in the design space.

It can be inferred from the convergence plots, that a significant improvement in the average fitness value is noted with generations. Furthermore, the distance of the best designs with generations converges to a minimum which is expected. Furthermore, the algorithm seems to handle the discretized design variables with ease and without any complications. Hence, the implementation of the genetic algorithm proves to be robust and shows promise in dealing with our complex problem.

Hyperparameters

There are many hyperparameters for the genetic algorithm which need to be experimentally determined and tweaked to get the best performance for the specified problem. Specifically, the population size, number of generations and mutation rate are some of the crucial parameters which are to be decided. Starting from the population size, as pointed out earlier, it is usually chosen to be set an order of magnitude higher than the number of design variables. In our case, it was determined via trial and error that a population size of 8000 individuals yielded the best diversity in the initial population pool. A population size greater than that resulted in many repeated designs. Hence, 8000 was chosen as the population size.

With regards to the number of generations, it was seen, yet again through trial and error that populations usually converged to an optimum after 50-60 generations. Hence, considering a margin of safety, all the optimizations were run for 70 generations.

Finally, the study on the effect of the mutation rate on convergence was conducted. Four different initial mutation rates were used to experiment. All of the algorithm runs used a dynamically changing mutation rate. Every 10 generations, the mutation rate was reduced by 80%. For the initial mutation rates of 50%, 25%, 10% and 1%, the convergence plots are shown in Figure 9.

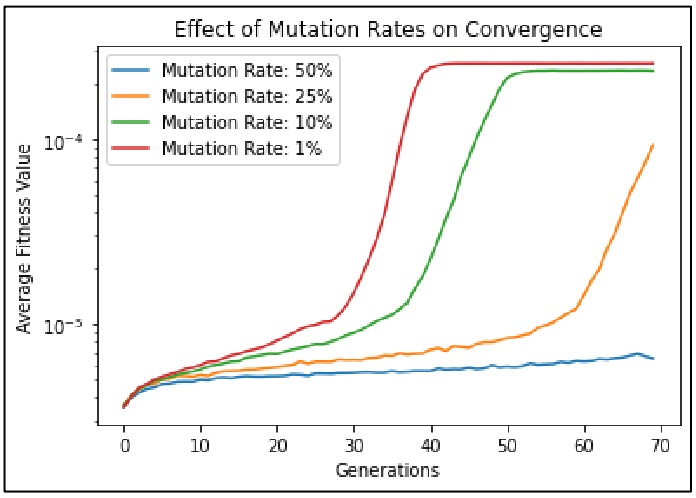

Figure 9: Effect of Mutation Rate on Convergence

It can be seen that lower initial mutation rates yielded the best improvement in average fitness values. Hence, an initial mutation rate of 15% was used in our implementation.

Main Problem Optimization

The optimization of the main problem was done in three scenarios. The first two scenarios tested the objective functions for minimizing distance travelled during a trip and for minimizing number of charging stops separately. The third and the final scenario tested the objective function for a combination of both the objectives. This was done to obtain a clearer understanding of the solutions. Furthermore, for each of these scenarios four test runs were used for finding the optimum to increase the possibility of capturing the optimal design since the initial population is randomized. Each test used 8000 individuals, 70 generations and varied initial mutation rates [15%, 10%, 5%, and 1%] to further improve the odds of finding the optimum.

Objective – Minimize Distance Travelled

The objective of minimizing distance travelled is first investigated. The modified objective function for this scenario if given by the following equation:

$$F(x,\ \mu_1,\ \mu_2) =$$

$$\frac{1}{\alpha \cdot \sum_{i=1}^{n}\left[\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\,\right] + C_{vio}(x,\ \mu_1,\ \mu_2)}$$

The minimum found for same is given by:

$$[x_i^* \,; y_j^*] = [4, 2, 7, 3, 5, 0, 6, 1\ ;\ 1, 0, 1, 0, 0, 0, 0]$$

with an optimal distance of $f(x_i^*) = 3341.64$ m. The convergence plots are plotted for this case below in Figure 10 and 11.

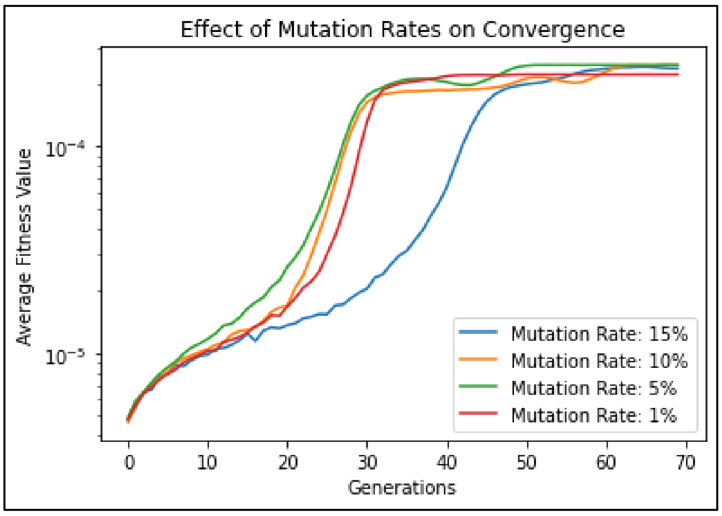

Figure 9: Convergence plot for Scenario 1

Here, it can be seen that the convergence rates for various mutation rates are identical for Scenario-1. Even though the same improvement on the average population fitness is noticed there is a major difference between the optimum designs each of the test runs reveal.

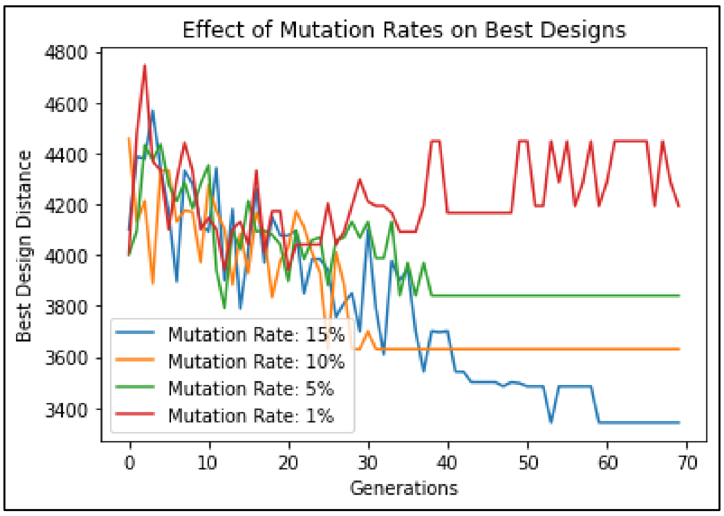

Figure 10: Minimum distance for best designs for Scenario 1

As shown in Figure 10, the minimum distance yielded by mutation rate of 15% is the best solution. Another interesting phenomenon noted here is that every this this test is run, the optimal changes as a result of the randomized initial population. This results in the algorithm, occasionally getting lucky/unlucky with some really good/poor designs.

Objective – Minimize Charging Stops

Here, the objective of minimizing the number of charging stop was investigated. The modified objective function for this scenario if given by the following equation:

$$F(x,\ \mu_1,\ \mu_2) = \frac{1}{\beta \cdot [\#\ of\ charging\ stops] + C_{vio}(x,\ \mu_1,\ \mu_2)}$$

The optimization found the following optimum design:

$$[x_i^* \, ; y_j^*] = [7, 2, 5, 0, 6, 1, 3, 4 \, ; \; 0, 1, 0, 0, 0, 1, 0]$$

which only uses two charging stops which the minimum required stops to make any trip in this design space feasible. The convergence plot for the same is shown in Figure 11. It can be seen that the initial mutation rate of 15%, yields slightly better performance for the average fitness for the final generation.

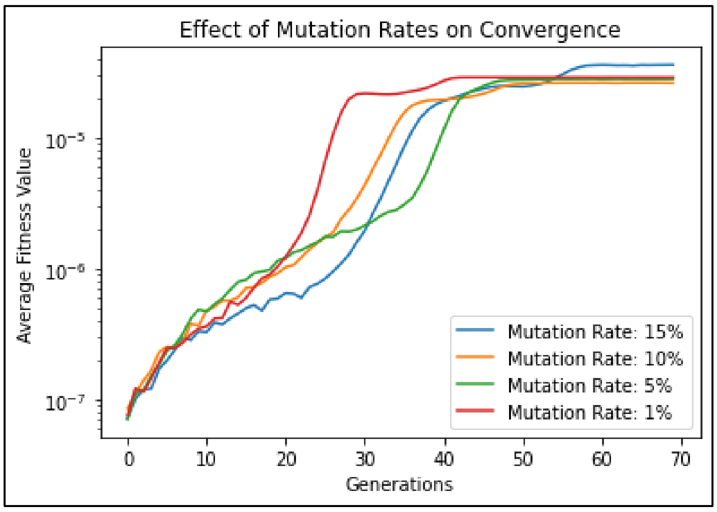

Figure 11: Convergence plot for Scenario 2

Furthermore, a rather interesting observation can be made when the total distance for the best design solutions is plotted for all the scenarios which is shown in Figure 12.

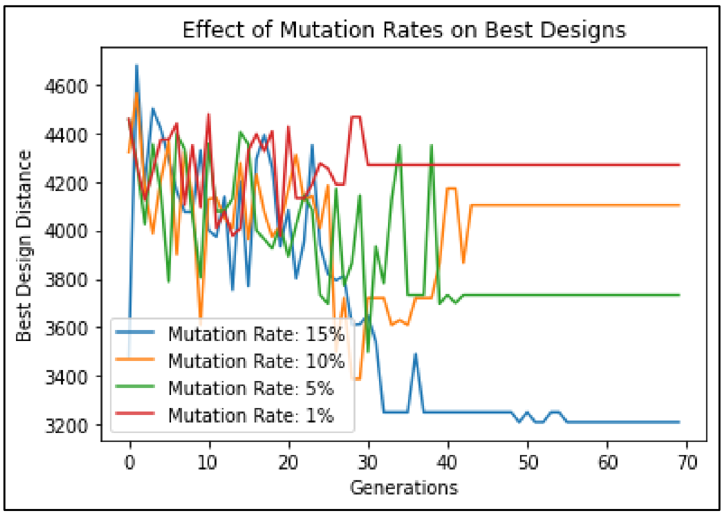

Figure 12: Minimum distance for best designs for Scenario 2

It can be noted that the best design for the initial mutation rate of 15% yields a distance travelled of $f(x_i^*) = 3210.28$ m. This is interesting because when the objective was to simply minimize the distance travelled in Scenario-1, the solution found had a greater distance than this. Consequently, this implies and verifies our belief about the coupling of the both the objectives and hence, if we were to find a minimum then that objective function shall include both the terms.

Objective – Minimize both objectives

Finally, for Scenario-3, the case of minimizing both the distance travelled and the charging stops, the objective function remains the same as the one in the problem formulation. The optimization found the following optimum design:

$$[x_i^* \,; y_j^*] = [7,3,4,2,5,0,6,1\,;\ 1,0,1,0,0,0,0]$$

resulting in a minimum distance of $f(x_i^*) = 3251.29$ m with only 2 charging stops. The convergence plot for the same is shown in Figure 13. It can be seen that the initial mutation rate of 1%, yields slightly better performance for the average fitness for the final generation.

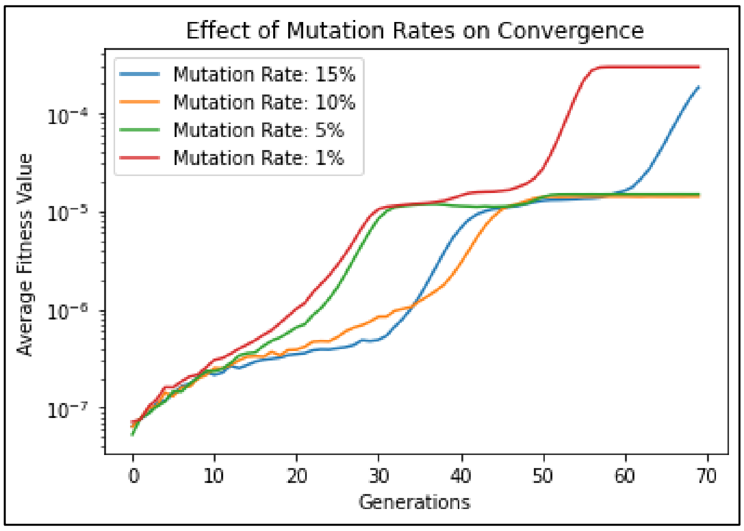

Figure 13: Convergence plot for Scenario 3

Even though an initial mutation rate of 1% yielded a better average fitness for the final generation, the run with an initial mutation rate of 10% also gave comparable minimum distances.

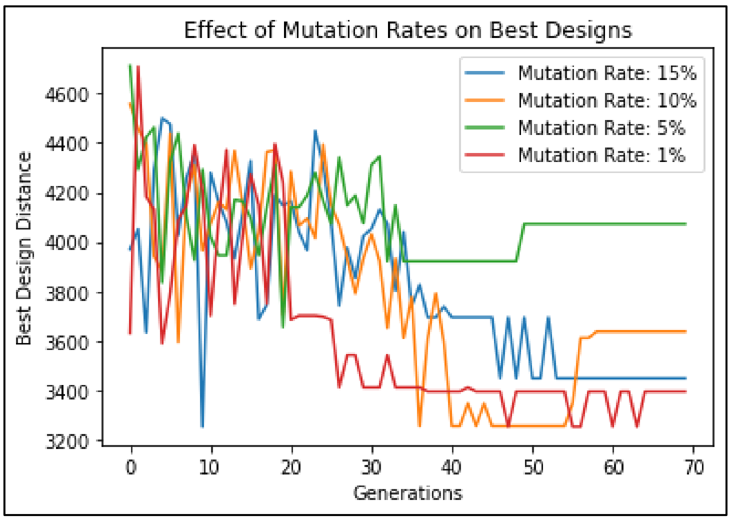

Figure 14: Minimum distance for best designs for Scenario 3

Upon conducting further runs, with an initial mutation rate of 15% , a slightly better optimum design was found. This design is given below :

$$[x_i^* \,; y_j^*] = [7,3,4,2,1,6,0,5\,;\ 1,0,1,0,0,0,0]$$

resulting in a minimum distance of $f(x_i^*) =\ 3055.73$ m with only 2 charging stops. This design is only slightly different from the optimal design found in Scenario-3.

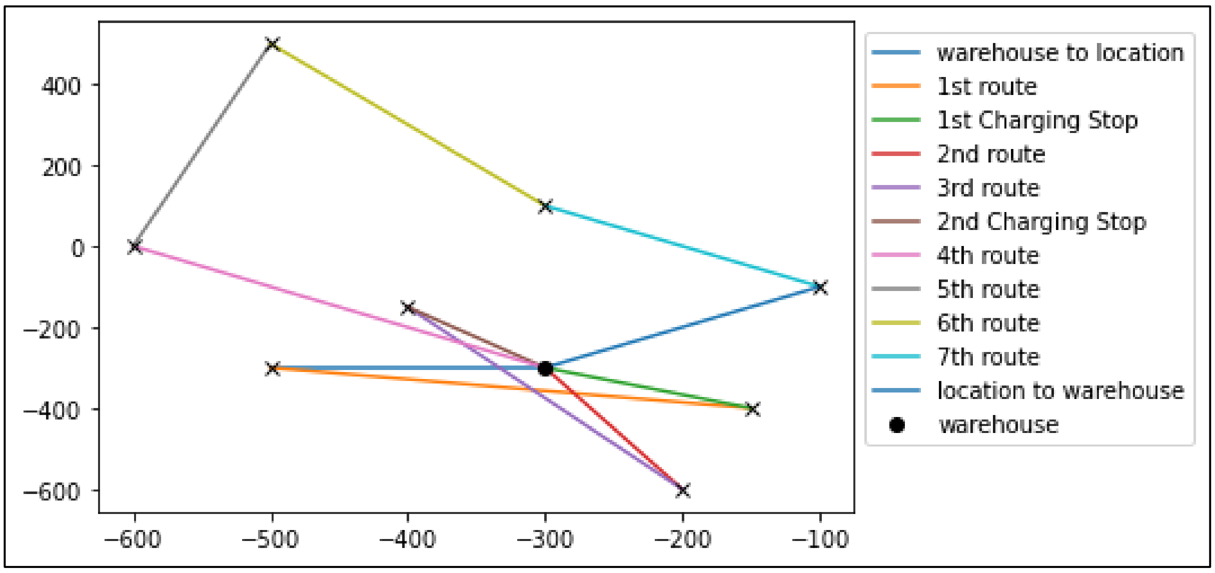

Figure 15: Optimal Route for delivery with charging stops

The entire route which minimizes both the distance travelled during a trip and the charging stops is plotted below in Figure 15. It can be verified visually that an optimal route was found by the algorithm.

Conclusion and Recommendations

It can be seen from all the scenarios that the genetic algorithm is successful in finding the optimum values. This method proves to be fast, efficient and robust for problems with large design space and discrete variables. Even though this method is successful in finding the minima, it is still incapable of yielding consistent results every run. This is due to its randomized initialization of populations. Sometimes, the algorithm gets 'lucky' and converges on a good design and sometimes it does not. Furthermore, a lot of experimentation has to be done in order to find the best hyperparameters for the genetic algorithm. Different problems have different optimum hyperparameters and the selection needs to be done carefully. Genetic algorithm proves to be valuable in getting important insights in sequential/route optimization.

This methodology of finding the optimum route for a drone delivery system proves to be extremely applicable to the last-mile-delivery operations. This is indeed a work in progress and a lot more work has to be done in terms of making this method efficient.

Future work can include the addition of delivery-time constraints, payload limitations for drones, accurate battery usage models, etc. which can make this more practical and applicable to real-world scenarios.

Lessons Learned

There was a myriad of lessons learned during this project which include the following but are not limited to:

1. it is very useful to understand the math behind the algorithms and why something ‘works’ as it helps in reasoning out and explaining the results from an optimization.
2. creating a structured plan for the algorithm by creating a pseudo-code before jumping into the implementation phase helps in keeping code clear and concise

Some factors where a lot of improvement could be made is to handle problems/issues/bugs in the code in a structured way rather than brute-forcing and trying to fix things instantly. A lot of time was spent doing this and a lot of time would be saved if improved upon.

References

[1]: https://www.emarketer.com/content/global-ecommerce-2019

[2]: Some images and pseudo algorithm code was also used from Multidisciplinary Design Optimization – Joaquim Martins, Andrew Ning