Walmart Delivery Drone Simulation

Felix Ruble, Mary Taylor, Megan Taylor

Motivation

Background

- Rapid growth of e-commerce[1]
  - Accelerated by Coronavirus[2]
- Challenge of last mile delivery
  - Costly and complicated part of supply chain[3]
- Emergence of Amazon Prime Air
  - Amazon received permission from FAA to operate as a drone airline on 8/31/2020[4]
- Drone delivery is rapidly approaching implementation

Enterprise

Drones are used in many applications:

- Photography
- Infrastructure inspection
- Agricultural support
- Package delivery

All have different needs of the drones

UAS Integration Pilot Program

- Generate communications between the government and the private sector on the development and integration of drone operations into the national airspace system[5].
- Participants of the program are selected under many criteria[6].
- Potential for using a simulation of drone operations to evaluate concept.
- **A simulation allows better understanding of the operations**
- Require Drone Operation Certifications

Drone Operation Certifications

- Title 14 of the Code of Federal Regulations (14 CFR) part 135 certificates
- Standard Part 135[7]: “does not have pre-set limits on the available size or scope of their operations”
- Several companies have received certifications from the FAA:

Company	Date	Certification Type	Purpose
Wing Aviation, LLC	October 2019	Standard Part 135 air carrier certificate	Deliver food and over the counter pharmaceuticals to homes in Christiansburg, VA
UPS FLight Forward, Inc	September 27, 2019	Standard Part 135 air carrier certificate	supplying medical supplies to a retirement community[8]
Amazon Prime Air	August 2020	Standard Part 135 air carrier certificate	deliver packages to low density areas with packages under 5 pounds[9]

Drone Specifications: Current Examples

- Three constraints: size, weight, and power.
- Amazon Drones:
    - weight limit is 5 lbs - 86% of amazon orders[10]
    - within 10 miles of the Amazon fulfilment center[11]
    - fly up to 15 miles[12]
- UPS Drones:
    - M2 Drone by Matternet
    - lift 2 kg and 4 liters (4.41 lbs and 244 cubic inches)
    - distance of up to 20 km[13] (12.43 miles)

Drone Specifications: Batteries

Specs: 22000mAh 6S 22.2V 15C LiPo battery

Discharge Rate:

Discharge Time = (Battery Capacity - Current Charge) / Average Amp Draw

- Discharge Rate Depends on Load and Amp Draw
- This battery can last about 32 min with a light load.

Charge Rate:

Charge Time = (Battery Capacity - Current Charge) / Charge Current

- Full charge (at 1 times the battery capacity) in 1 hour

Simulated Problem Description

Problem and Need Statement

Problem	Need
There is much criteria to meet when applying for a drone delivery system.	Develop a simulation to determine the criteria that can be used when applying for a drone delivery system.
Drone Delivery system has not been created full scale.	Develop a simulation to compare different configurations in their ability to complete deliveries within reasonable delivery times and cost estimates.
Competitors, like Amazon, already have a test drone delivery system in place.	We would like to simulate a hypothetical drone delivery system launched by Walmart, using Fairfax Walmart as a use case

Simulated Problem Description

- Model a drone delivery service at the Fairfax Walmart
  - 15 census tracts
  - 5 miles radius
- Transient simulation, each run is a different day
- Goal: compare different number of drones and chargers used for different proportions of total demand
- Three different proportions of total demand used, not expected that all retail demand will be delivered through drones (10%, 33%, and 50%).
- Outputs: total deliver time/wait time, energy usage, and energy cost.

Simulation Configurations

Configuration Number	Proportion of Total Demand	Number of Drones	Number of Chargers
1	10%	7	6
2		8	5
3		8	7
4		12	8
5		12	9
6	33%	24	16
7		24	21
8		25	20
9		26	21
10		29	22
11	50%	37	24
12		37	25
13		38	27
14		38	32
15		42	31

Input Data Analysis and Model Assumptions

Key Model Parameters and Inputs - Drones

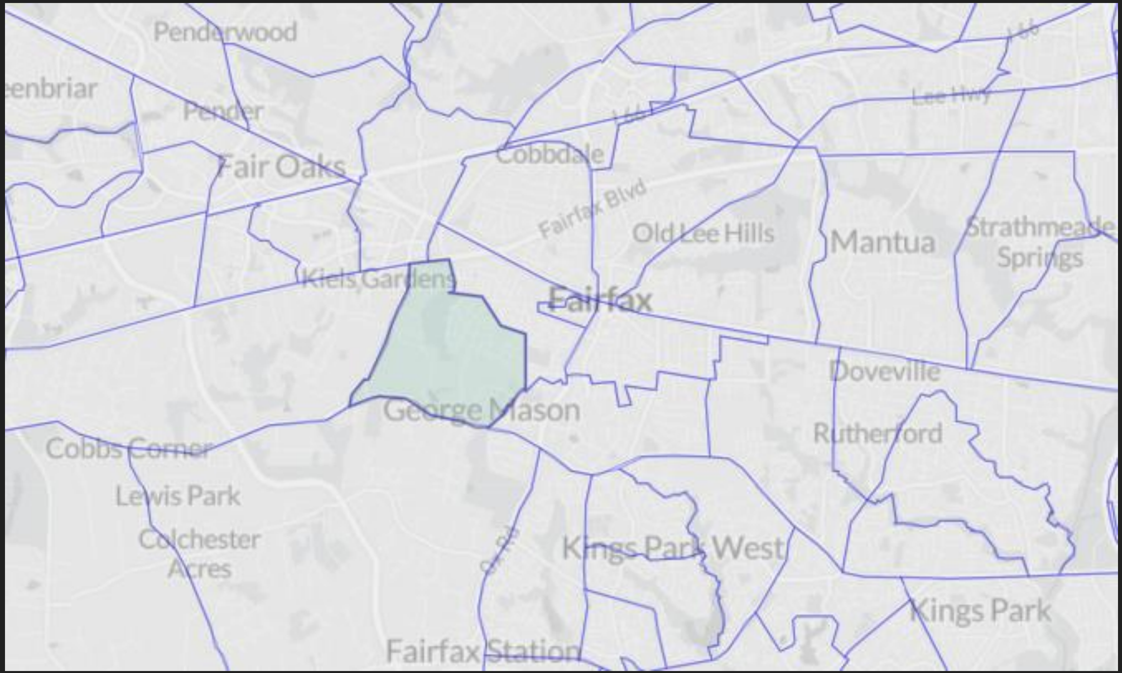
- Speed (50 mph)
- Battery Capacity (22000 mAh)
- Time to Charge (1 hour - max)
- Distance Flown (radius) (5.4 miles max)
- Number of Drones (listed in configuration chart)
- Number of Chargers (listed in configuration chart)

Key Model Parameters and Inputs - Demand

- To model orders we need estimates for order arrival rates and distance between order destinations and our hub
- Estimating general demand for orders
  - Using market research on Amazon and Walmart to approximate parameters for online delivery
  - Amazon gets 5.8 % of retail spending, Walmart gets 8.9 % [14]
  - Average household spends 31 % of after tax income on retail [15]
  - Average Walmart purchase is $19-30, Average Amazon purchase is $36-60 [16]
- Use Census data to estimate population and incomes
  - Break down area around Walmart store into set of census tracts
  - Each census tract generates orders

Key Model Parameters and Inputs - Demand

- Break area around Walmart down by census tract
- Use income and population data to estimate an upper bound for order arrival rate for each census tract
- Randomly generate order inter-arrival times for each census tract using exponential distribution



Key Model Parameters and Inputs - Order Distance

- Each order has an associated distance
- Using the census tract map, measure:
  - Minimum and maximum distance from Walmart for each census tract
  - Distance from population center of census tract to Walmart
- Randomly generate order distance from triangular distribution using minimum, centroid and maximum distance for each census tract as min, mode and max in a triangular distribution

Census Tract Data

Census Tract	Min Dist	Max Dist	Mode Dist	# Households	Median Income	Post Tax Income	Orders per HH per Year	Orders per Year	Orders per Day
4406	0.25	1.5	0.65	938	118571	83000	37	34995	96
3004	0.25	1.5	0.84	1364	108977	76284	34	46771	128
3002	1.35	3.3	2.12	1723	137313	96119	43	74443	204
3001	0.5	1.55	0.94	1874	94063	65844	30	55465	152
4612.02	0.25	1.7	0.73	2583	117588	82312	37	95569	262
4905.02	0.25	4.5	2.47	2003	188295	131807	59	118672	325
4917.03	0.8	1.8	1.12	2310	106836	74785	34	77653	213
4917.04	1.35	2.25	1.77	2109	97574	68302	31	64750	177
4817.02	1.25	2.8	2.06	1934	107577	75304	34	65464	179
3003	1.6	3.45	2.58	2356	99119	69383	31	73478	201
4619.01	2.5	3.35	3.04	1353	101382	70967	32	43160	118
4616.01	3.15	5.35	4.12	2827	110673	77471	35	98445	270
4617	3.5	5.4	4.12	2334	103611	72528	33	76091	208
4401	3.35	5	3.85	2540	163188	114232	51	130422	357
4618.02	1.25	2.7	2.23	2358	115474	80832	36	85675	235

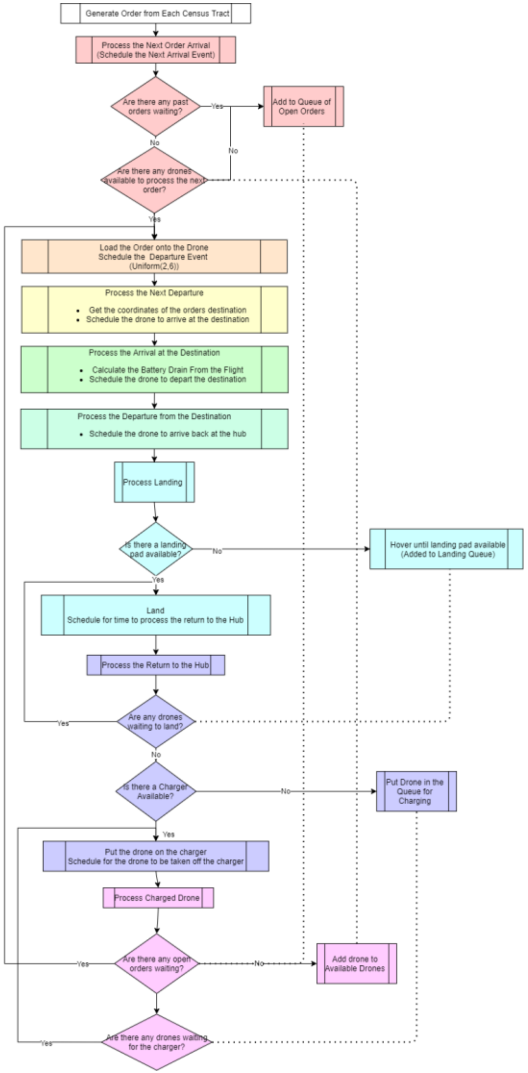
Model Assumptions

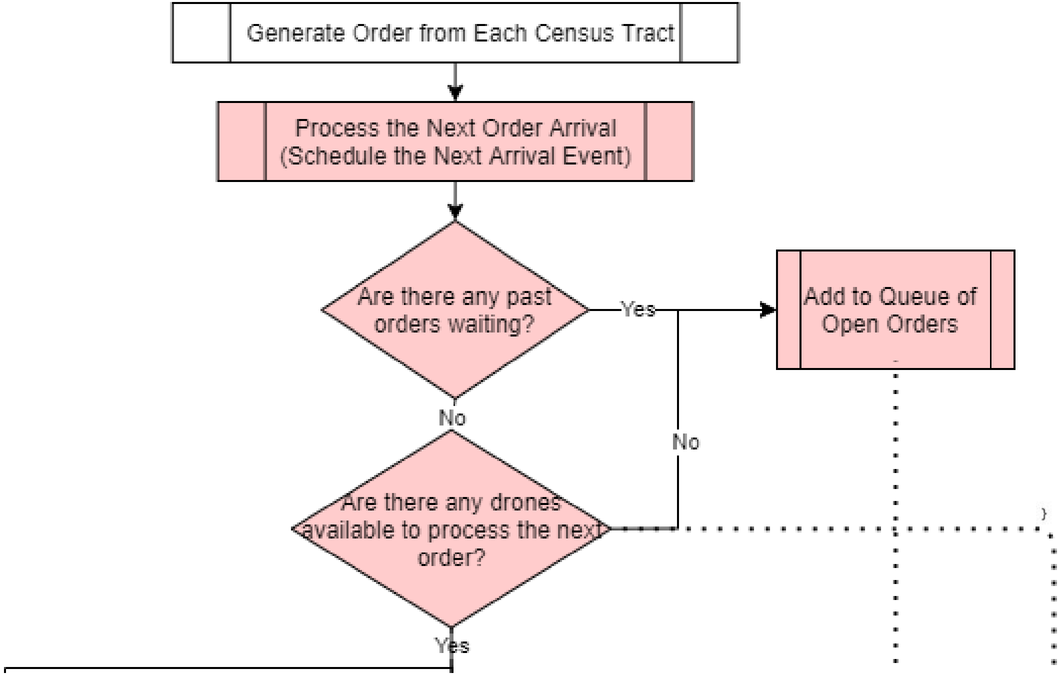
- All drones use the same battery
- Perfect weather conditions
- Drone flies at 50 mph
- Battery specs used: 22000mAh 6S 22.2V 15C LiPo battery
- Average Amp Draw:
  - With Package: 60 A
  - Without Package: 20 A
- Currently no failures

Simulation Model/Codes

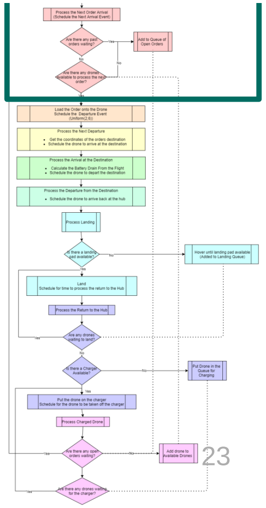
Simulation Flow Chart

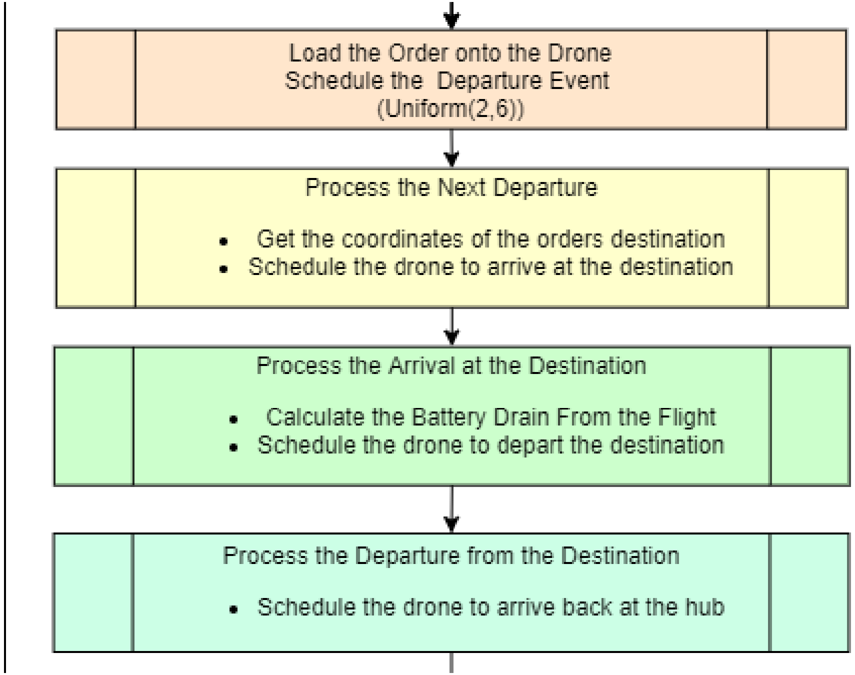
Adding and
removing from
queue/inventory



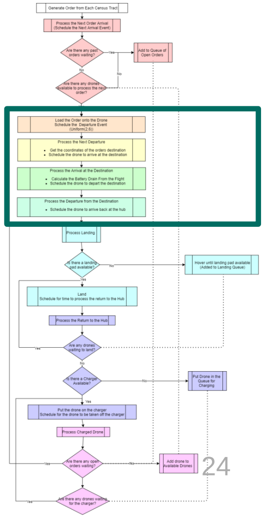


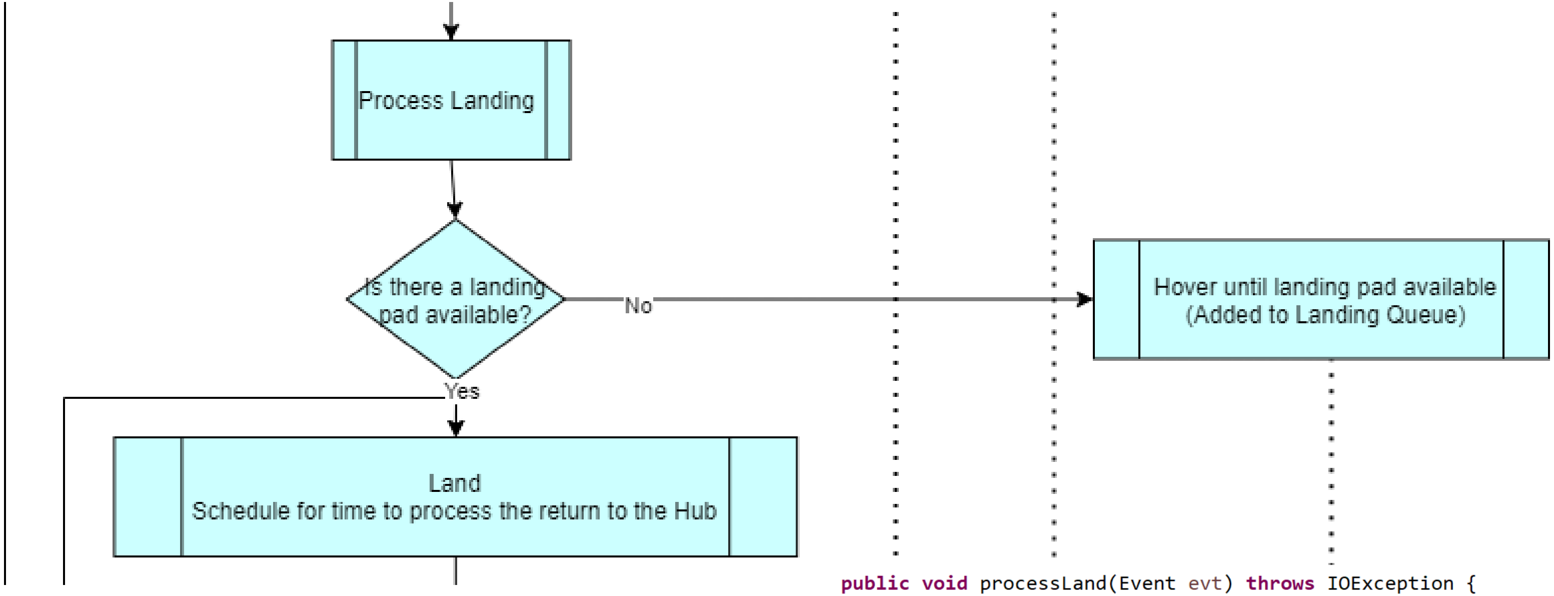

```java
public void ProcessNodeOrder(String tractName) throws Exception {
    // a drone has just arrived
    if (Clock % 1440 > 480 && Clock % 1440 < 1200) {
        usingFileWriter(Integer.toString(hub.getOrderQueueLength()), "OutputFiles/OrderQueueLength.txt", Clock,
                tractName, (Clock % 1440) / 60); // update
        // info
        Double distance = demandNodes.get(tractName).generateOrderDistance(stream);
        Order order = new Order(distance);// create the order object
        order.setTract(tractName);
        order.setTimePlaced(Clock);
        // are there any orders in the queue
        if (hub.orderAvailable()) {
            hub.addOrder(order);
        }
        // There were no orders in the queue
        else {
            // where there any drones available
            if (hub.droneAvailable()) {
                hub.addOrder(order);
                hub.load();
                double time = uniform(stream, 2, 6);
                Event departure = new Event(depart, Clock + time);// set the time to a loading time (stochastic)
                departure.setDrone(order.getDrone());
                FutureEventList.enqueue(departure);

            }
            // There were no drones available
            else {
                hub.addOrder(order);
            }
        }
    }
    ScheduleNodeOrder(tractName);// schedules the next order
}
```



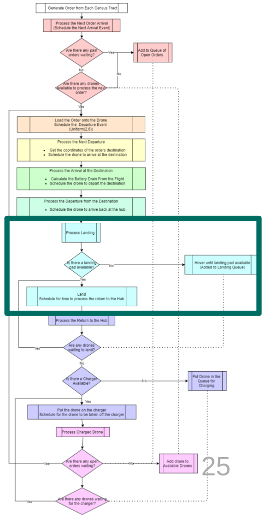



```java
public void leaveDestination(Event evt) throws IOException {
    usingFileWriter(Integer.toString(numOrders + 1), "OutputFiles/OrderFulfillment.txt", Clock,
            evt.getDrone().getOrder().getTract(), (Clock % 1440) / 60);
    Drone drone = evt.getDrone();
    Double flightTime = Clock - drone.getDepartTime();
    // System.out.println(numOrders + "\t" + flightTime);
    drone.getOrder().setTimeReceived(Clock);
    drone.setBattery(drone.getBattery() - (flightTime / 60) * 60);
    Double TripTime = drone.getOrder().getTimeRecieved() - drone.getOrder().getTimePlaced();
    usingFileWriter(Double.toString(TripTime), "OutputFiles/OrderTime.txt", Clock,
            evt.getDrone().getOrder().getTract(), (Clock % 1440) / 60);
    drone.setDepartTime(Clock);
    numOrders += 1;
    totalTime += TripTime;
    // scheduling when it returns from the trip
    double unload = uniform(stream, 1, 5);
    double time = drone.travel(drone.getOrder().getDistance());
    Event Trip = new Event(needland, Clock + time + unload);
    Trip.setDrone(drone);
    FutureEventList.enqueue(Trip);

}
```

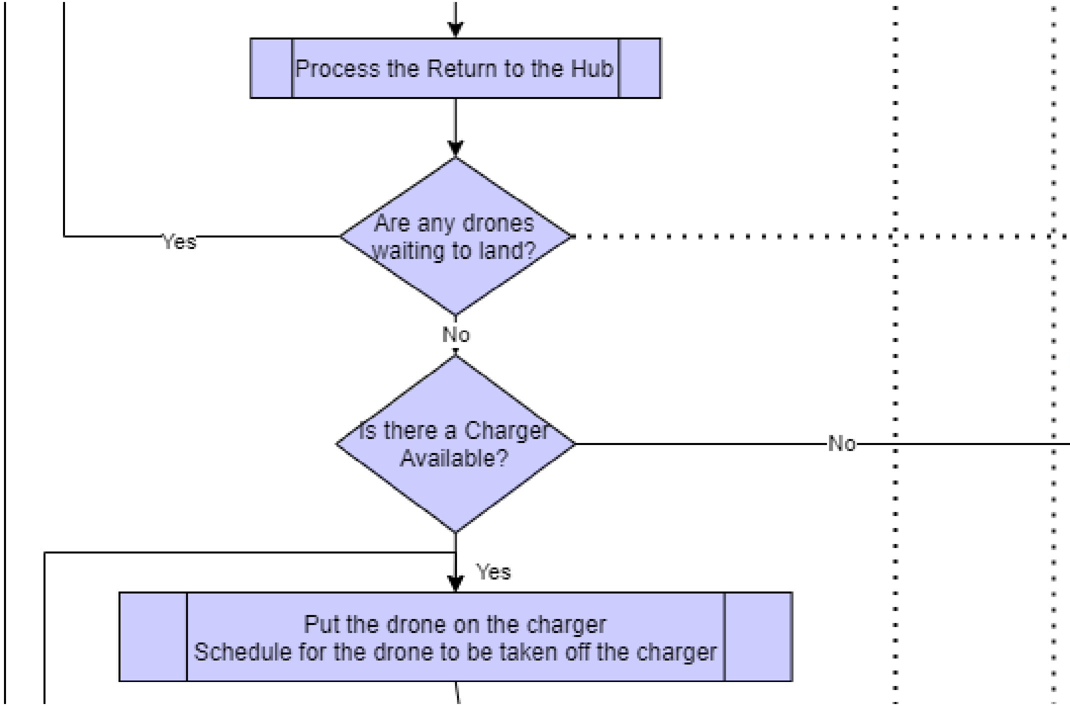


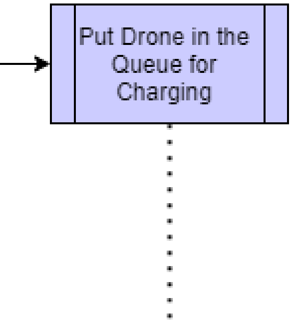


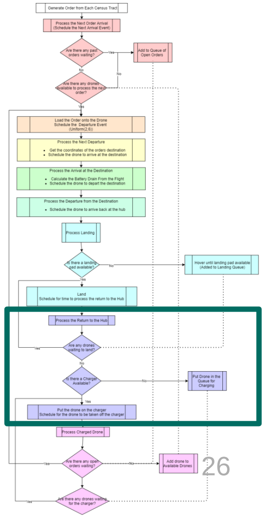

```java
public void processLand(Event evt) throws IOException {
    Drone drone = evt.getDrone();
    if (hub.landingAvailable()) {
        hub.Land(drone);
        double time = uniform(stream, 1, 3);
        Event Trip = new Event(trip, Clock + time);
        Trip.setDrone(drone);
        FutureEventList.enqueue(Trip);
    } else {
        hub.AddtoLandingQueue(drone);
    }
}
```



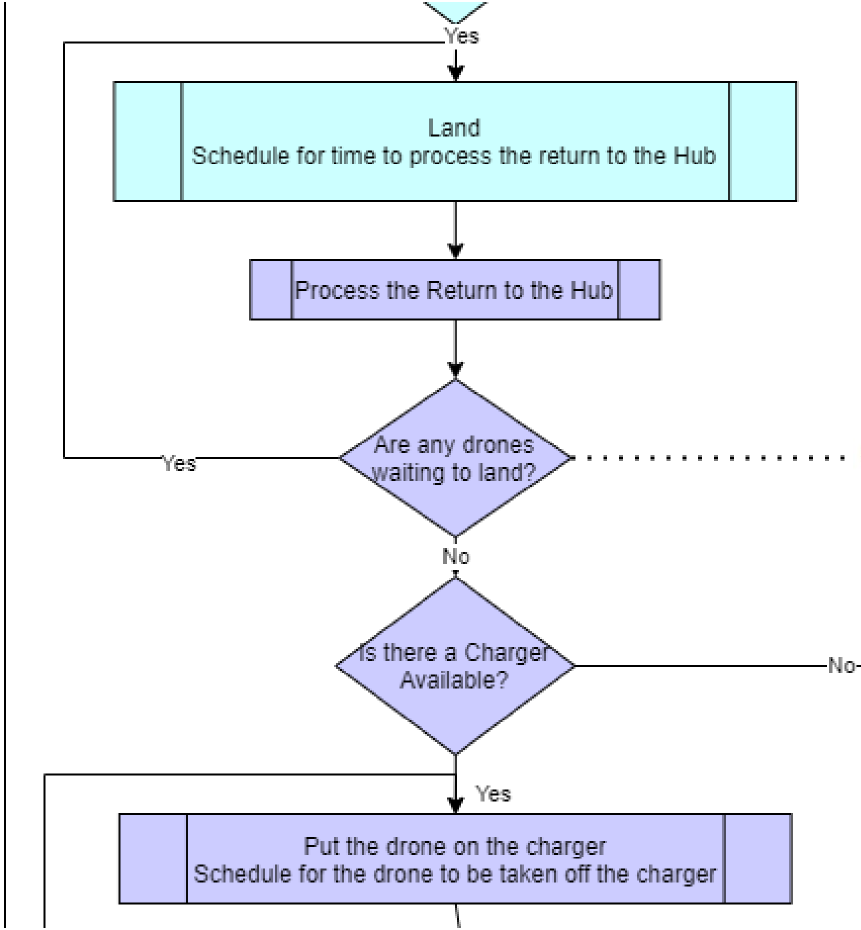
```java
public void processTrip(Event evt) throws IOException {
    Double flightTime = Clock - evt.getDrone().getDepartTime();
    Drone drone = evt.getDrone();
    evt.getDrone().setBattery(evt.getDrone().getBattery() - (flightTime / 60) * 20);
    hub.removefromLandingPad(drone);
    if (hub.droneNeedLand()) {
        Drone drone2 = hub.getNextDroneLand();
        processLand(drone2);
    }

    // schedule add to charging queue
    // check if there is a charger available
    if (hub.chargerAvailable()) {
        hub.chargeDrone(drone);
        double time = ((battery.getAh() - drone.getBattery()) / battery.getAh() * battery.getChargeC() * 60);

        Event DroneCharged = new Event(droneCharged, Clock + time); // time based on equation
        usingFileWriter(Double.toString(drone.getBattery()), "OutputFiles/BatteryLevel.txt", Clock,
                drone.getOrder().getTract(), (Clock % 1440) / 60);
        DroneCharged.setDrone(drone);
        FutureEventList.enqueue(DroneCharged);
    } else {
        hub.addDronetoCharge(drone, Clock);
    }
}
```

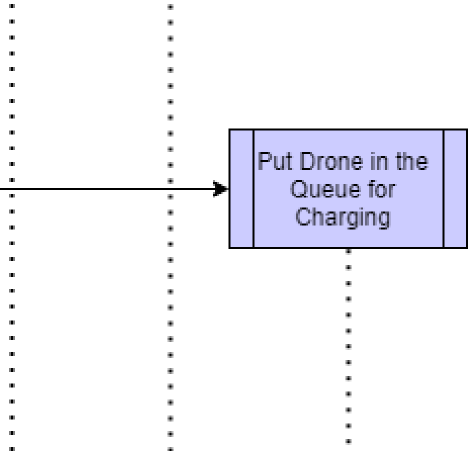


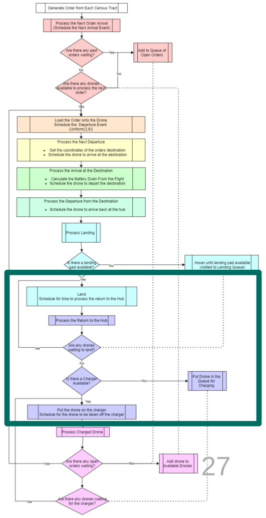


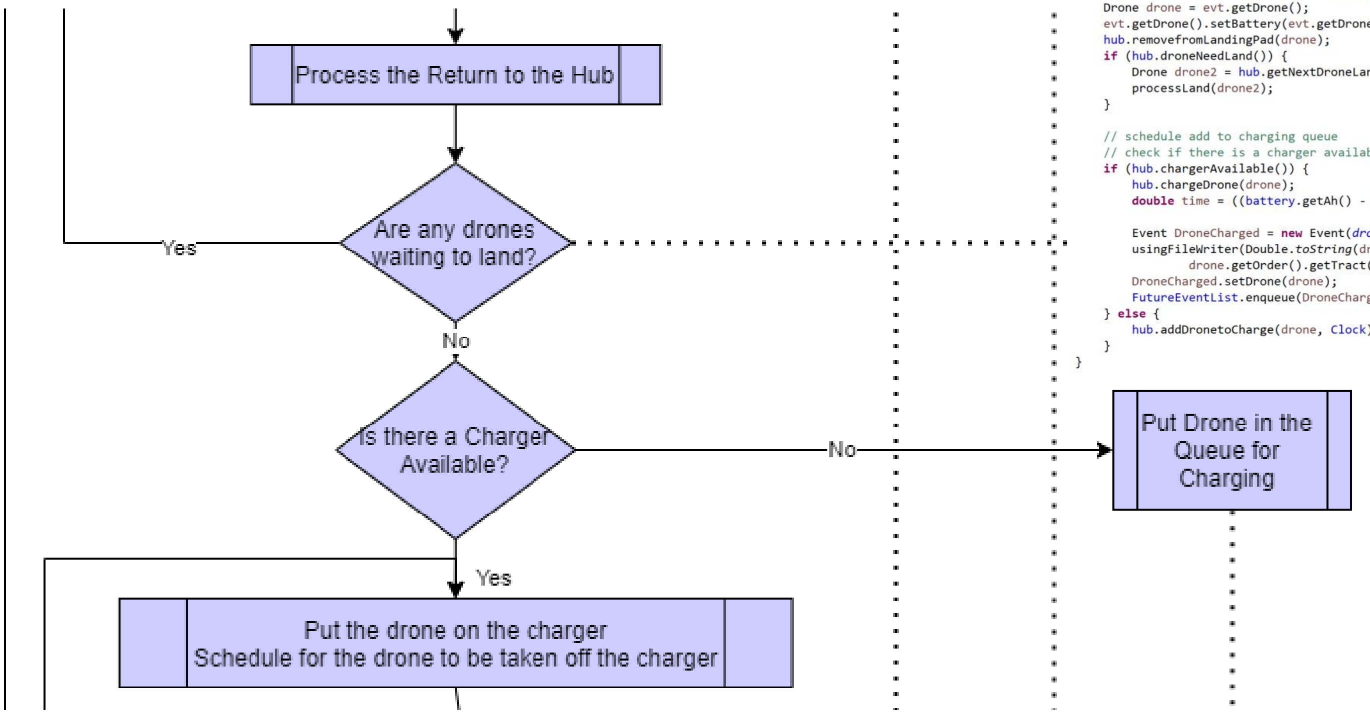


```java
public void processTrip(Event evt) throws IOException {
    Double flightTime = Clock - evt.getDrone().getDepartTime();
    Drone drone = evt.getDrone();
    evt.getDrone().setBattery(evt.getDrone().getBattery() - (flightTime / 60) * 20);
    hub.removefromLandingPad(drone);
    if (hub.droneNeedLand()) {
        Drone drone2 = hub.getNextDroneLand();
        processLand(drone2);
    }

    // schedule add to charging queue
    // check if there is a charger available
    if (hub.chargerAvailable()) {
        hub.chargeDrone(drone);
        double time = ((battery.getAh() - drone.getBattery()) / battery.getAh() * battery.getChargeC() * 60);

        Event DroneCharged = new Event(droneCharged, Clock + time); // time based on equation
        usingFileWriter(Double.toString(drone.getBattery()), "OutputFiles/BatteryLevel.txt", Clock,
                drone.getOrder().getTract(), (Clock % 1440) / 60);
        DroneCharged.setDrone(drone);
        FutureEventList.enqueue(DroneCharged);
    } else {
        hub.addDronetoCharge(drone, Clock);
    }
}
```

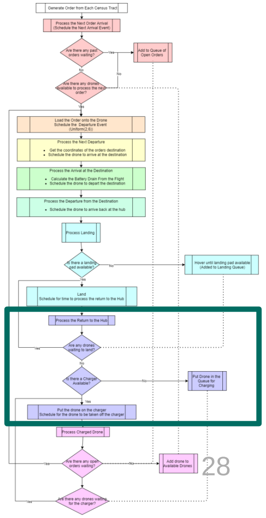


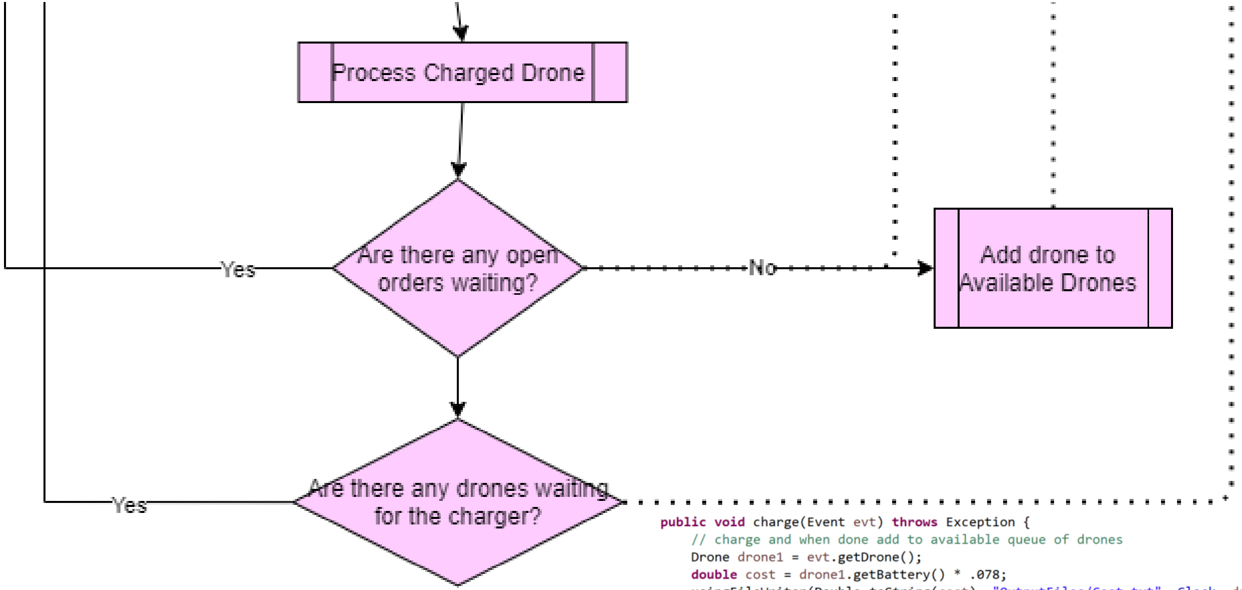




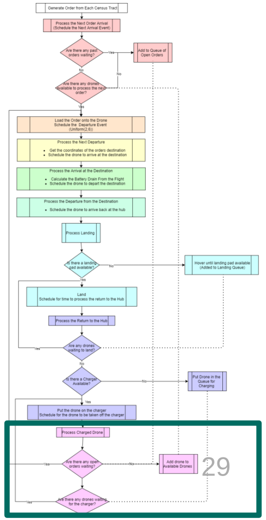


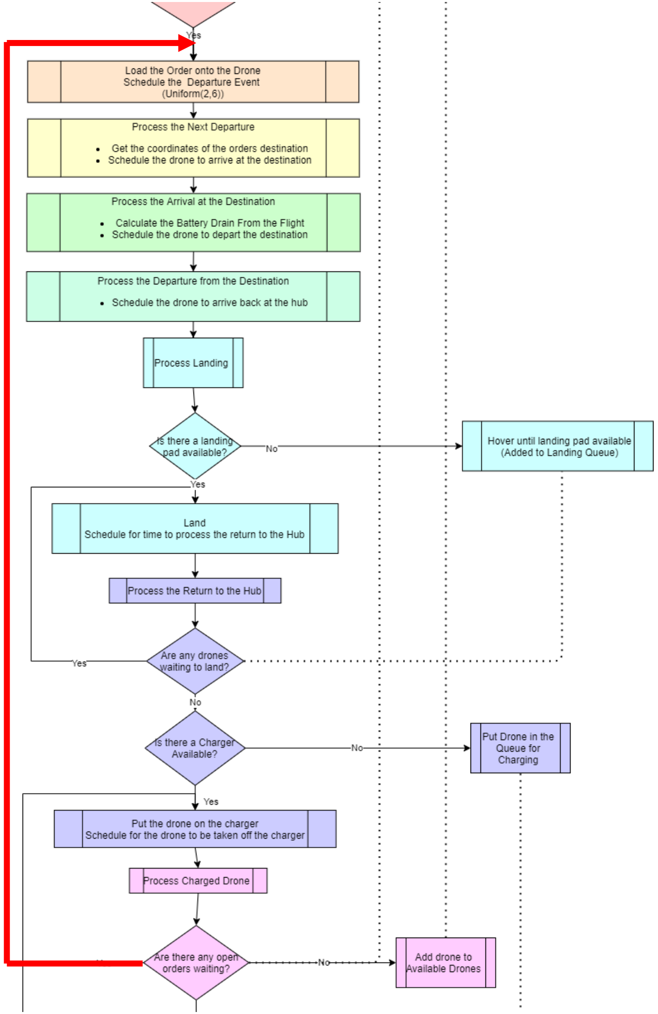

```java
public void processTrip(Event evt) throws IOException {
    Double flightTime = Clock - evt.getDrone().getDepartTime();
    Drone drone = evt.getDrone();
    evt.getDrone().setBattery(evt.getDrone().getBattery() - (flightTime / 60) * 20);
    hub.removefromLandingPad(drone);
    if (hub.droneNeedLand()) {
        Drone drone2 = hub.getNextDroneLand();
        processLand(drone2);
    }

    // schedule add to charging queue
    // check if there is a charger available
    if (hub.chargerAvailable()) {
        hub.chargeDrone(drone);
        double time = ((battery.getAh() - drone.getBattery()) / battery.getAh() * battery.getChargeC() * 60);

        Event DroneCharged = new Event(droneCharged, Clock + time); // time based on equation
        usingFileWriter(Double.toString(drone.getBattery()), "OutputFiles/BatteryLevel.txt", Clock,
                drone.getOrder().getTract(), (Clock % 1440) / 60);
        DroneCharged.setDrone(drone);
        FutureEventList.enqueue(DroneCharged);
    } else {
        hub.addDronetoCharge(drone, Clock);
    }
}
```



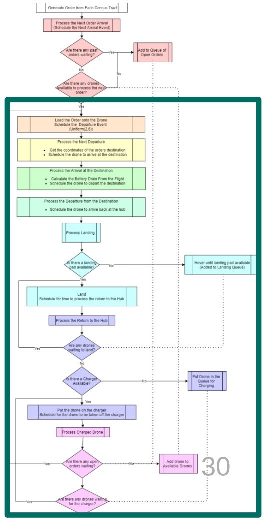


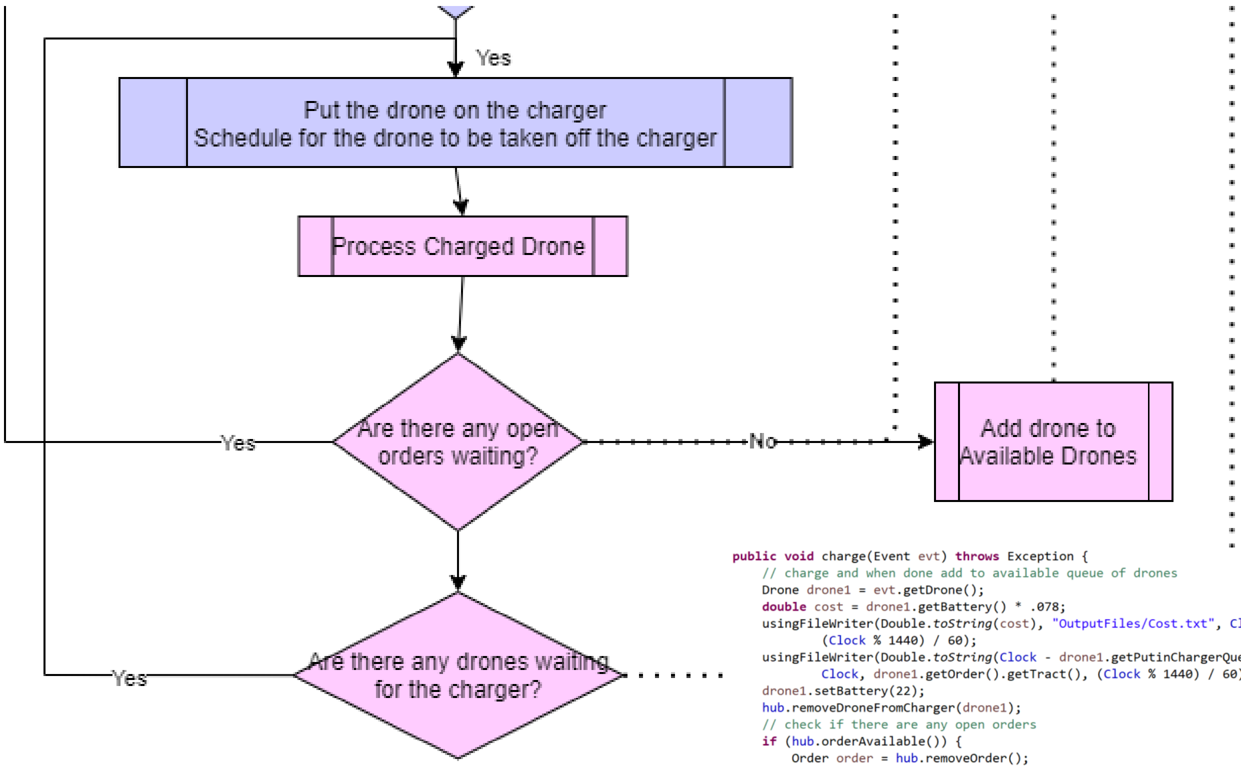

```
public void charge(Event evt) throws Exception {
    // charge and when done add to available queue of drones
    Drone drone1 = evt.getDrone();
    double cost = drone1.getBattery() * .078;
    usingFileWriter(Double.toString(cost), "OutputFiles/Cost.txt", Clock, drone1.getOrder().getTract(),
            (Clock % 1440) / 60);
    usingFileWriter(Double.toString(Clock - drone1.getPutinChargerQueue()), "OutputFiles/ChargerQueueLength.txt",
            Clock, drone1.getOrder().getTract(), (Clock % 1440) / 60);
    drone1.setBattery(22);
    hub.removeDroneFromCharger(drone1);
    // check if there are any open orders
    if (hub.orderAvailable()) {
        Order order = hub.removeOrder();
        ProcessNodeOrder(order.getTract(), order, drone1);
    } else {
        hub.addDrone(drone1);
    }

    if (hub.droneWaitingToCharge()) {
        Drone drone = hub.getNextDrone();
        hub.chargeDrone(drone);
        double time = ((battery.getAh() - drone.getBattery()) / battery.getAh() * battery.getChargeC() * 60);

        Event DroneCharged = new Event(droneCharged, Clock + time); // time based on equation
        DroneCharged.setDrone(evt.getDrone());
        FutureEventList.enqueue(DroneCharged);
    }

}
```



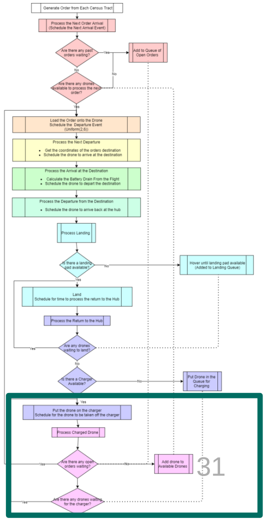



```java
public void charge(Event evt) throws Exception {
    // charge and when done add to available queue of drones
    Drone drone1 = evt.getDrone();
    double cost = drone1.getBattery() * .078;
    usingFileWriter(Double.toString(cost), "OutputFiles/Cost.txt", Clock, drone1.getOrder().getTract(),
            (Clock % 1440) / 60);
    usingFileWriter(Double.toString(Clock - drone1.getPutinChargerQueue()), "OutputFiles/ChargerQueueLength.txt",
            Clock, drone1.getOrder().getTract(), (Clock % 1440) / 60);
    drone1.setBattery(22);
    hub.removeDroneFromCharger(drone1);
    // check if there are any open orders
    if (hub.orderAvailable()) {
        Order order = hub.removeOrder();
        ProcessNodeOrder(order.getTract(), order, drone1);
    } else {
        hub.addDrone(drone1);
    }

    if (hub.droneWaitingToCharge()) {
        Drone drone = hub.getNextDrone();
        hub.chargeDrone(drone);
        double time = ((battery.getAh() - drone.getBattery()) / battery.getAh() * battery.getChargeC() * 60);

        Event DroneCharged = new Event(droneCharged, Clock + time); // time based on equation
        DroneCharged.setDrone(evt.getDrone());
        FutureEventList.enqueue(DroneCharged);
    }

}
```






```java
public void charge(Event evt) throws Exception {
    // charge and when done add to available queue of drones
    Drone drone1 = evt.getDrone();
    double cost = drone1.getBattery() * .078;
    usingFileWriter(Double.toString(cost), "OutputFiles/Cost.txt", Clock, drone1.getOrder().getTract(),
            (Clock % 1440) / 60);
    usingFileWriter(Double.toString(Clock - drone1.getPutinChargerQueue()), "OutputFiles/ChargerQueueLength.txt",
            Clock, drone1.getOrder().getTract(), (Clock % 1440) / 60);
    drone1.setBattery(22);
    hub.removeDroneFromCharger(drone1);
    // check if there are any open orders
    if (hub.orderAvailable()) {
        Order order = hub.removeOrder();
        ProcessNodeOrder(order.getTract(), order, drone1);
    } else {
        hub.addDrone(drone1);
    }

    if (hub.droneWaitingToCharge()) {
        Drone drone = hub.getNextDrone();
        hub.chargeDrone(drone);
        double time = ((battery.getAh() - drone.getBattery()) / battery.getAh() * battery.getChargeC() * 60);

        Event DroneCharged = new Event(droneCharged, Clock + time); // time based on equation
        DroneCharged.setDrone(evt.getDrone());
        FutureEventList.enqueue(DroneCharged);
    }

}
```



Output Statistical Analysis

Determining Number of Replications - 10% of Demand

$$n(\beta) = \min\left\{n : t_{n-1,0.0.95}\sqrt{\frac{718.991}{n}} \leq \beta\right\}, \beta = 2.0$$

$$n = 100 : t_{n-1,0.0.95}\sqrt{\frac{718.991}{n}} \approx 1.98422\sqrt{\frac{718.991}{100}} \approx 5.320$$

$$n = 700 : t_{n-1,0.0.95}\sqrt{\frac{718.991}{n}} \approx 1.9634\sqrt{\frac{718.991}{700}} \approx 1.990$$

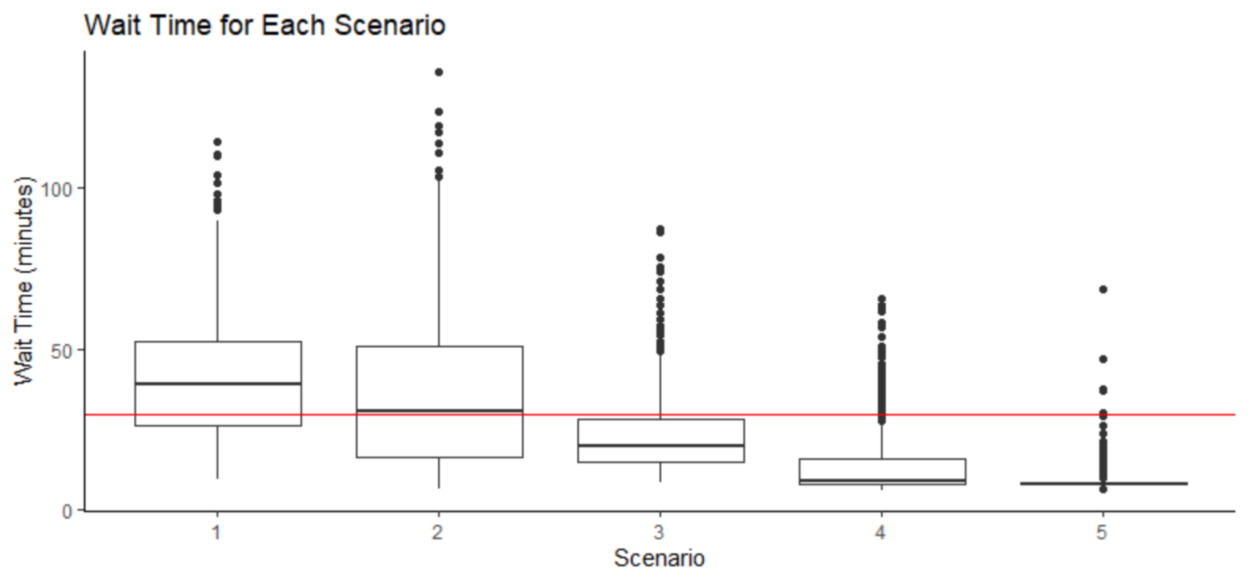
$$n = 700 : t_{n-1,0.0.95}\sqrt{\frac{564.728}{n}} \approx 1.96473\sqrt{\frac{564.728}{700}} \approx 1.7635$$

Absolute Error, alpha = 0.05

Outputs – 10% Demand

Scenario	Number of Chargers	Number of Drones	Wait Time	WaitTime.sd	energy	energy.sd	cost	cost.sd
1	6	7	41.63	19.45	0.16	0.04	132.09	36.11
2	5	8	36.03	23.76	0.17	0.04	59.52	19.19
3	7	8	23.39	12.07	0.16	0.03	170.31	25.47
4	8	12	14.47	11.13	0.16	0.04	153.16	34.33
5	9	12	8.98	3.90	0.16	0.03	182.78	19.39

Comparing Wait Time for Scenarios – 10% Demand



Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
1	10%	7	6
2		8	5
3		8	7
4		12	8
5		12	9

Comparing Wait Time for Scenarios – 10% Demand

Pairwise comparison of wait time with 95% confidence: each confidence interval has 99.5% confidence

	2	3	4	5
1	[2.376233 , 8.819007]	[15.81663 , 20.650176]	[24.738699 , 29.572499]	[30.576556 , 34.720036]
2		[9.769174 , 15.502392]	[18.724305 , 24.391653]	[24.50055 , 29.600802]
3			[7.185998 , 10.658394]	[13.079199 , 15.750587]
4				[4.266598 , 6.718796]

Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
1	10%	7	6
2		8	5
3		8	7
4		12	8
5		12	9

Configuration 1 has the greatest wait time, while configuration 5 has the shortest wait time. Wait time compares across configurations as follows: $time_1 > time_2 > time_3 > time_4 > time_5$

Comparing Estimated Cost for Scenarios – 10% Demand

Estimating cost using $0.078 per kWh for electricity, $1500 per drone, $500 per charger, we produce the following pairwise comparison of cost across possible configurations

	2	3	4	5
1	[-931.6087 , -923.261]	[-2042.8478 , -2033.6078]	[-8526.306 , -8515.8491]	[-9055.1617 , -9046.225]
2		[-1114.2264 , -1107.3596]	[-7597.8116 , -7589.4738]	[-8126.1594 , -8120.3576]
3			[-6487.5329 , -6478.1665]	[-7015.9431 , -7008.988]
4				[-533.7313 , -525.5004]

Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
1	10%	7	6
2		8	5
3		8	7
4		12	8
5		12	9

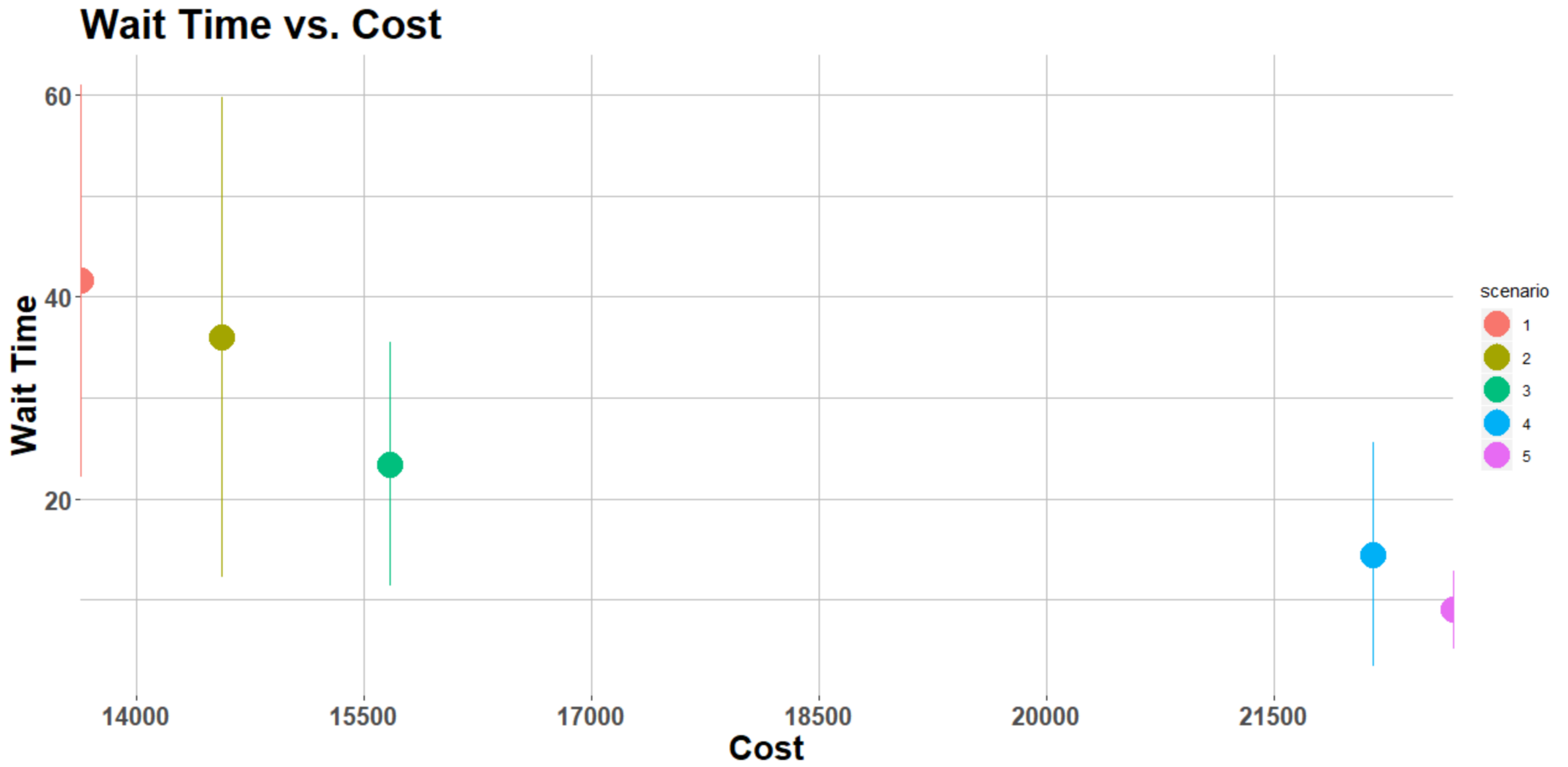
Configuration 1 has the lowest cost, while configuration 5 has the greatest cost. Estimated cost compares across configurations as follows: $cost_5 > cost_4 > cost_3 > cost_2 > cost_1$

Comparing Wait Time for Scenarios – 10% Demand

ANOVA

	1	2	3	4	5
1	NA	NA	NA	NA	NA
2	2.9e-09	NA	NA	NA	NA
3	2.6e-09	2.6e-09	NA	NA	NA
4	2.6e-09	2.6e-09	2.6e-09	NA	NA
5	2.6e-09	2.6e-09	2.6e-09	3.2e-09	NA

Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
1	10%	7	6
2		8	5
3		8	7
4		12	8
5		12	9


Wait Time vs. Cost
60
40
20
Wait Time
14000
15500
17000
18500
20000
21500
Cost
scenario
1
2
3
4
5

Determining Number of Replications - 33% of Demand

$$n(\beta) = \min\left\{n: t_{n-1,0.0.95}\sqrt{\frac{429.982}{n}} \leq \beta\right\}, \beta = 2.0$$

$$n = 100: t_{n-1,0.0.95}\sqrt{\frac{429.982}{n}} \approx 1.9842\sqrt{\frac{429.982}{100}} \approx 4.114$$

$$n = 419: t_{n-1,0.0.95}\sqrt{\frac{429.982}{n}} \approx 1.9657\sqrt{\frac{429.982}{419}} \approx 1.991$$

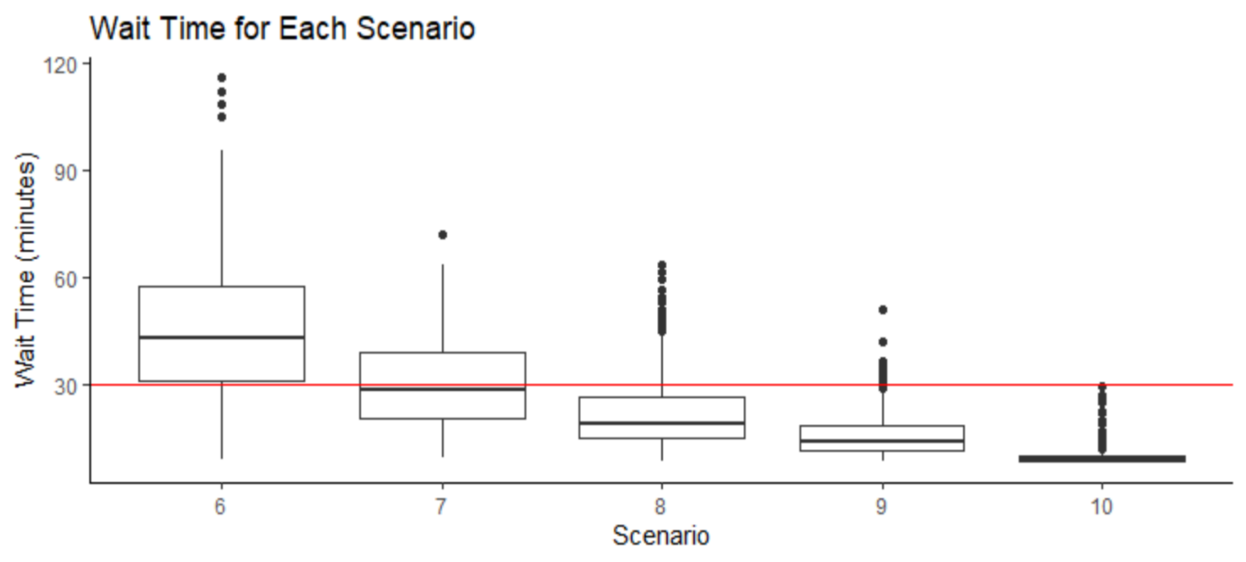
$$n = 450: t_{n-1,0.0.95}\sqrt{\frac{368.602}{n}} \approx 1.9653\sqrt{\frac{368.602}{450}} \approx 1.7787$$

Absolute Error,
alpha = 0.05

Outputs – 33% Demand

Scenario	Number of Chargers	Number of Drones	Wait Time	WaitTime.sd	energy	energy.sd	cost	cost.sd
6	16	24	45.49	19.20	0.18	0.04	389.83	85.70
7	21	24	30.01	12.34	0.15	0.03	575.95	16.08
8	20	25	21.88	9.95	0.15	0.03	594.05	23.80
9	21	26	15.75	6.10	0.15	0.03	601.59	22.12
10	22	29	9.65	2.46	0.15	0.03	612.41	33.09

Comparing Wait Time for Scenarios – 33% Demand



Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
6	33%	24	16
7		24	21
8		25	20
9		26	21
10		29	22

Comparing Wait Time for Scenarios – 33% Demand

Pairwise comparison of wait time with 95% confidence: each confidence interval has 99.5% confidence

	7	8	9	10
6	[12.528637 , 18.434697]	[20.676612 , 26.53843]	[27.081648 , 32.393515]	[33.276058 , 38.412376]
7		[5.966974 , 10.284734]	[12.367937 , 16.143892]	[18.678329 , 22.046771]
8			[4.651476 , 7.608646]	[10.869653 , 13.603739]
9				[5.234954 , 6.978316]

Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
6	33%	24	16
7		24	21
8		25	20
9		26	21
10		29	22

Configuration 6 has the greatest wait time, while configuration 10 has the shortest wait time. Wait time compares across configurations as follows: $time_6 > time_7 > time_8 > time_9 > time_{10}$

Comparing Estimated Cost for Scenarios – 33% Demand

Estimating cost using $0.078 per kWh for electricity, $1500 per drone, $500 per charger, we produce the following pairwise comparison of cost across possible configurations

	7	8	9	10
6	[-931.6087 , -923.261]	[-2042.8478 , -2033.6078]	[-8526.306 , -8515.8491]	[-9055.1617 , -9046.225]
7		[-1114.2264 , -1107.3596]	[-7597.8116 , -7589.4738]	[-8126.1594 , -8120.3576]
8			[-6487.5329 , -6478.1665]	[-7015.9431 , -7008.988]
9				[-533.7313 , -525.5004]

Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
6	33%	24	16
7		24	21
8		25	20
9		26	21
10		29	22

Configuration 6 has the lowest cost, while configuration 10 has the greatest cost. Estimated cost compares across configurations as follows: $cost_{10} > cost_9 > cost_8 > cost_7 > cost_6$

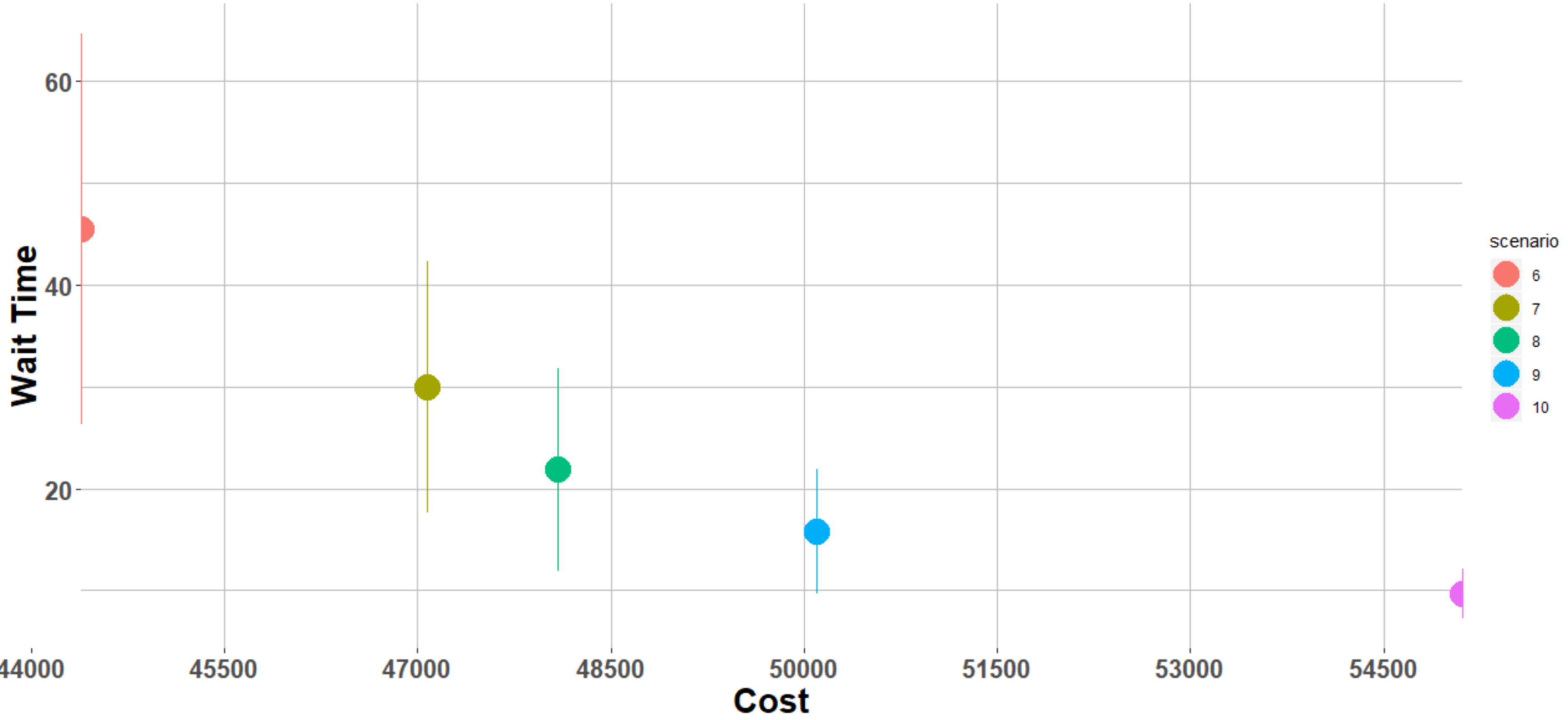
Comparing Wait Time for Scenarios – 33% Demand

ANOVA

	6	7	8	9	10
6	NA	NA	NA	NA	NA
7	1e-10	NA	NA	NA	NA
8	1e-10	1e-10	NA	NA	NA
9	1e-10	1e-10	1e-10	NA	NA
10	1e-10	1e-10	1e-10	1e-10	NA

Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
6	33%	24	16
7		24	21
8		25	20
9		26	21
10		29	22

Wait Time vs. Cost



Determining Number of Replications - 50% of Demand

$$n(\beta) = \min\left\{ n: t_{n-1,0.0.95}\sqrt{\frac{294.706}{n}} \le \beta \right\}, \beta = 2.0$$

$$n = 100: t_{n-1,0.0.95}\sqrt{\frac{294.706}{n}} \approx 1.98422\sqrt{\frac{294.706}{100}} \approx 3.406$$

$$n = 288: t_{n-1,0.0.95}\sqrt{\frac{294.706}{n}} \approx 1.96826\sqrt{\frac{294.706}{288}} \approx 1.9904$$

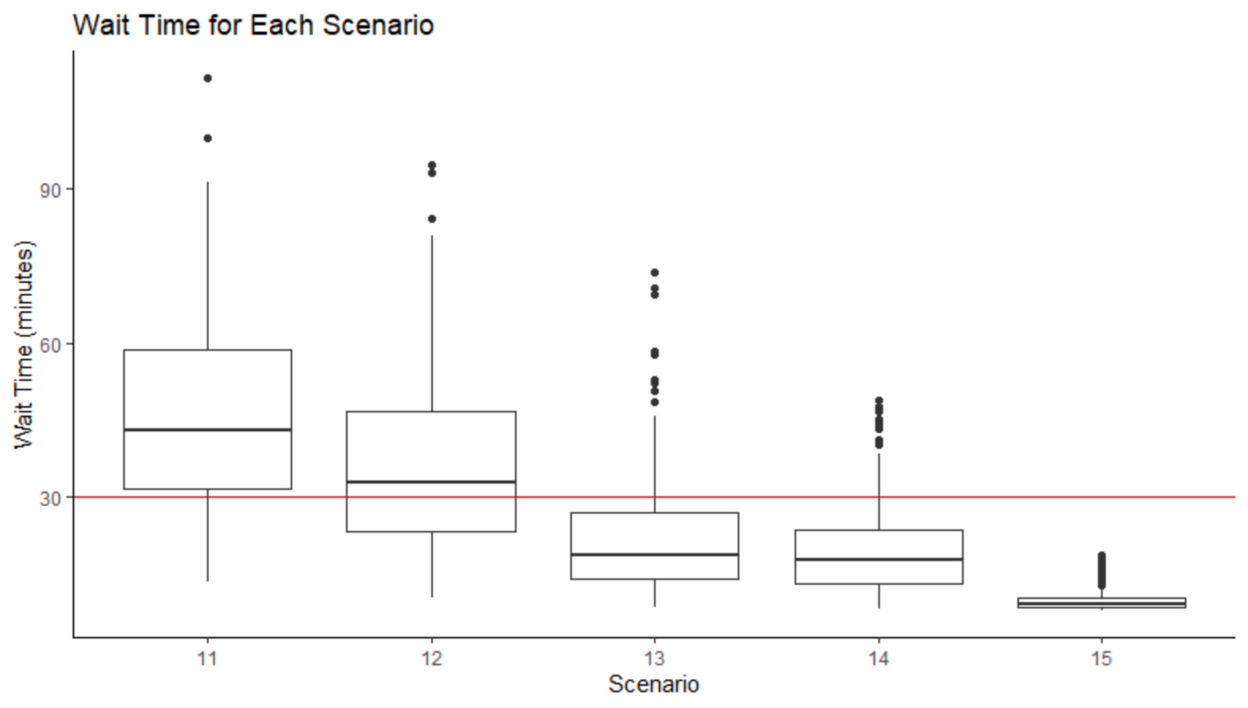
$$n = 300: t_{n-1,0.0.95}\sqrt{\frac{330.912}{n}} \approx 1.9679\sqrt{\frac{330.912}{300}} \approx 2.0668$$

Absolute Error,
alpha = 0.05

Outputs – 50% Demand

Scenario	Number of Chargers	Number of Drones	Wait Time	WaitTime.sd	energy	energy.sd	cost	cost.sd
11	24	37	45.38	18.19	0.18	0.05	753.75	125.05
12	25	37	35.69	15.95	0.17	0.04	809.90	119.70
13	27	38	22.03	11.38	0.16	0.04	898.67	78.36
14	32	38	19.55	8.19	0.15	0.03	900.63	23.67
15	31	42	9.82	1.82	0.15	0.03	925.32	34.66

Comparing Wait Time for Scenarios – 50% Demand



Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
11	50%	37	24
12		37	25
13		38	27
14		38	32
15		42	31

Comparing Wait Time for Scenarios – 50% Demand

Pairwise comparison of wait time with 95% confidence:
each confidence interval has 99.5% confidence

	12	13	14	15
11	[5.8485622 , 13.537784]	[19.9999052 , 26.712765]	[22.5803288 , 29.091107]	[32.555426 , 38.57581]
12		[10.4443664 , 16.881958]	[13.0870838 , 19.198006]	[23.247054 , 28.497836]
13			[0.1924341 , 4.766332]	[10.3366422 , 14.081924]
14				[8.3980948 , 11.061705]

Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
11	50%	37	24
12		37	25
13		38	27
14		38	32
15		42	31

Configuration 11 has the greatest wait time, while configuration 15 has the shortest wait time. Wait time compares across configuraitons as follows: $time_{11} > time_{12} > time_{13} > time_{14} > time_{15}$

Comparing Estimated Cost for Scenarios – 50% Demand

Estimating cost using $0.078 per kWh for electricity, $1500 per drone, $500 per charger, we produce the following pairwise comparison of cost across possible configurations

	12	13	14	15
11	[-584.896 , -527.4027]	[-3169.532 , -3120.3044]	[-5667.318 , -5626.442]	[-11192.963 , -11150.1841]
12		[-2611.597 , -2565.941]	[-5110.552 , -5070.909]	[-10635.742 , -10595.1064]
13			[-2515.382 , -2488.5417]	[-8041.056 , -8012.2552]
14				[-5531.654 , -5517.7337]

Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
11	50%	37	24
12		37	25
13		38	27
14		38	32
15		42	31

Configuration 11 has the lowest cost, while configuration 15 has the greatest cost. Estimated cost compares across configurations as follows: $cost_{15} > cost_{14} > cost_{13} > cost_{12} > cost_{11}$

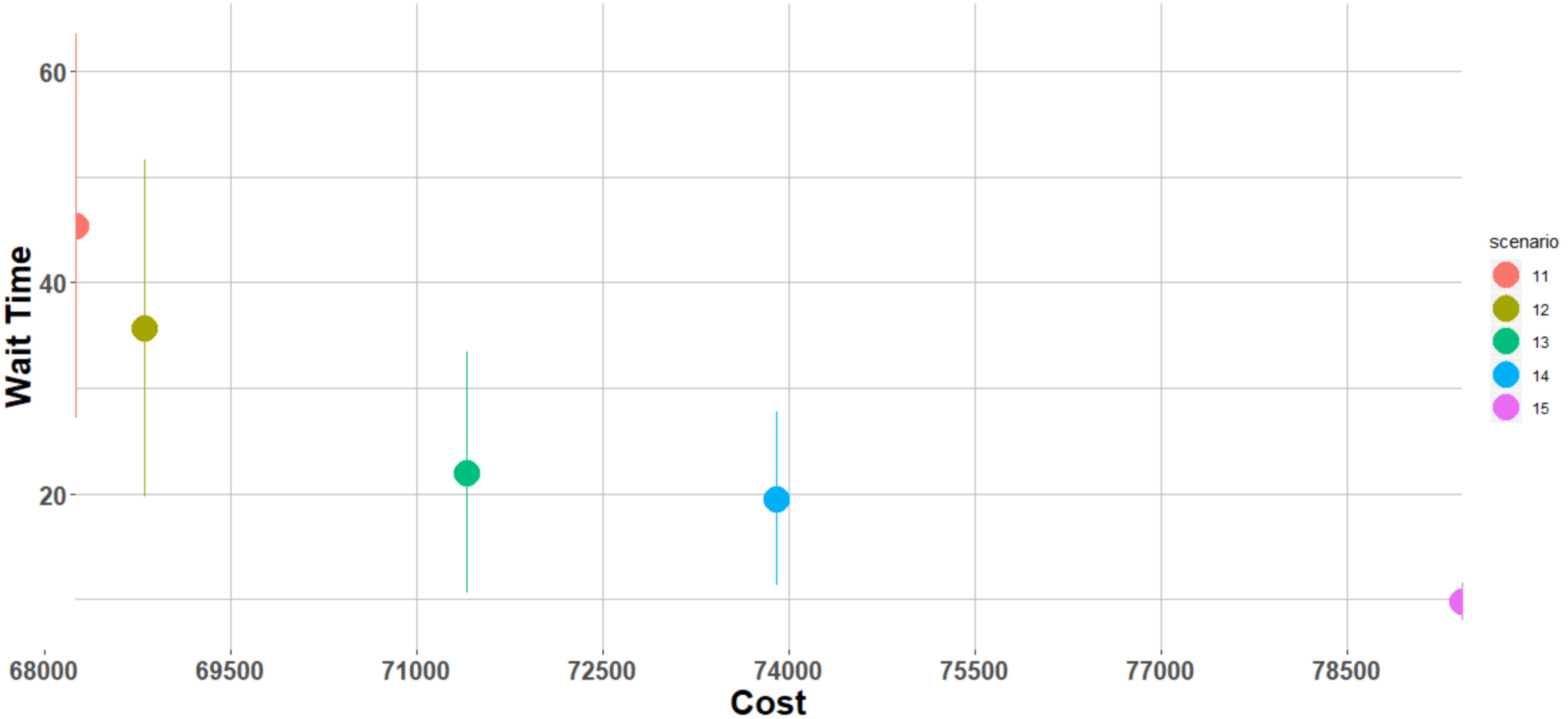
Comparing Wait Time for Scenarios – 50% Demand

ANOVA

	11	12	13	14	15
11	NA	NA	NA	NA	NA
12	0	NA	NA	NA	NA
13	0	0	NA	NA	NA
14	0	0	0.1096184	NA	NA
15	0	0	0.0000000	0	NA

Config. Number	Proportion of Total Demand	Number of Drones	Number of Chargers
11		37	24
12		37	25
13	50%	38	27
14		38	32
15		42	31

Wait Time vs. Cost



Extensions and Conclusions

Extensions

- Add non-stationary Poisson process if we had that data
- Determine if there is a relationship between the number of chargers and the number of drones regardless of the level of demand
- Sensitivity analysis can be conducted for the cost of the drones and chargers.
- With some modification, a fee for each delivery can be calculated.

Conclusions

- Our simulation shows that, assuming assumptions are reasonably accurate, it is feasible to implement an efficient drone delivery service at the Fairfax Walmart at a reasonable cost

- Which configuration to recommend is heavily dependent on how much demand is expected, and how long of a wait time is acceptable
  - wait time – cost charts should serve as a helpful guide to decision makers once the expected demand and acceptable wait time are determined

Sources

1. https://www.digitalcommerce360.com/article/us-ecommerce-sales/
2. https://www.forbes.com/sites/louiscolumbus/2020/04/28/how-covid-19-is-transforming-e-commerce/#568e3a373544
3. https://www.mckinsey.com/~/media/mckinsey/industries/travel%20transport%20and%20logistics/our%20insights/how%20customer%20demands%20are%20reshaping%20last%20mile%20delivery/parcel_delivery_the_future_of_last_mile.ashx
4. https://www.theverge.com/2020/8/31/21408646/amazon-prime-air-drone-delivery-faa-clearance-approval-health-safety-alphabet-wing

Sources

5. https://www.faa.gov/uas/programs_partnerships/integration_pilot_program/
6. https://www.whitehouse.gov/presidential-actions/presidential-memorandum-secretary-transportation/
7. https://www.faa.gov/licenses_certificates/airline_certification/135_certification/general_info/
8. https://www.theverge.com/2020/4/27/21238196/ups-cvs-drone-delivery-medicine-florida-coronavirus
9. https://www.cnbc.com/2020/08/31/amazon-prime-now-drone-delivery-fleet-gets-faa-approval.html

Sources

10. https://internetofthingsagenda.techtarget.com/definition/Amazon-Prime-Air-drone#:~:text=Amazon%27s%20Prime%20Air%20drone%20is%20a%20delivery%20system,with%20eight%20rotors%20--%20that%20use%20GPS%20navigation

11. https://www.dronetechplanet.com/what-drones-does-amazon-use

12. https://www.cnbc.com/2020/08/31/amazon-prime-now-drone-delivery-fleet-gets-faa-approval.html

13. https://mttr.net/product

Sources

14. https://www.pymnts.com/news/retail/2019/amazon-or-walmart-who-won-2019s-race-for-the-consumers-whole-paycheck/

15. https://www.pymnts.com/consumer-finance/2018/how-much-of-the-consumers-paycheck-goes-to-amazon/

16. https://www.zentail.com/blog/amazon-walmart-ebay-benchmarks-for-sellers