El Farol Bar Problem

Yi Liu - 1605486

In this Assignment, I was using python as programming language to implement the algorithm for solving El Farol Bar Problem. I have also tested my programme with different parameters to determine the impact of parameters on the efficiency of the population. For each parameter setting, I did 100 independent runs. I will describe my pseudo-code first, and then show you test data and plots.

Ex.4: Pseudo-Code

1. Initialise Population:

```
def initPop(popSize,h):
      i = 0
      popMatrix = []
      while(i < popSize):
              j = 0
              individual_without_H = []
              individual = []
              while(j < h):
                      a = [h random numbers that equals 1]
                      b = [h random numbers that equals 1]

                      state = state_value + a + b
                      individual_without_H.append(state)
                      j += 1
              individual.append(h)
              individual.append(individual_without_H)
              popMatrix.append(individual)
              i += 1
      return popMatrix
```

2. Mutation:

```
def mutation(individual,rate):
      h = individual[0]
      mutation_num = math.ceil(rate * h * 2)
      i = 0
      while(i < mutation_num):
              mutate_part = SELECT ONE PART TO MUTATE
              # SELECT MUTATE A OR B
              randAB = random.uniform(0,1)
              if randAB <= 0.5:
                      mutate_part[A] = [h random numbers that equal 1 in total]
              else:
                      mutate_part[B] = [h random numbers that equal 1 in total]
              mutate_part[0] = random.uniform(0,1)
              individual[MUTATE_PART] = mutate_part
              i += 1
      return individual
```

3. Crossover:

```
def crossover(individual1,individual2):
      # One part refers to one part of [Pi ai0..aih-1..bih-1]
      crossover_part = individual2[RANDOM SELECT ONE PART]
      individual1[RANDOM SELECT ONE PART] = crossover_part
      return individual1
```

4. Generation:

```
def generation(popSize,h,weeks,max_t):
     population = initPop(popSize,h)
     generationCount = 0
     while(generationCount < max_t):
             crowded = 0
             popCount = 0
             state = 0
             decision_list = []
             state_list = []
             # week loop
             payoff_list = [0 of len(popSize)]
             weekCount = 0
             while(weekCount < weeks):
                     new_decision_list = []
                     new_state_list = []
                     popCount = 0
                     while(popCount < popSize):
                             decision,new_state =
                                     SAMPLE THE DECISION AND THE STATE(EX2)
                             decision_list.append(decision)
                             state_list.append(new_state)
                             popCount += 1

                     # is it crowded?
                     peopleGoBar = NUMBER OF 1 IN decision_list
                     if(peopleGoBar > CROWDED_RATE * len(decision_list)):
                             crowded = 1
               payoff_list = PLUS 1 FOR PEOPLE STAYED HOME
           else:
             crowded = 0
             payoff_list = PLUS 1 FOR PEOPLE IN BAR
           printSet =
                 weekCount,generationCount,crowded,peopleGoBar,decision_list
           print(printSet)
           weekCount += 1
    rankPopSet = ranking(population,payoff_list)
    cutNum = int((1-CROWDED_RATE)*popSize)
    cutPopSet = rankPopSet[0:popSize-cutNum]
    addOffspring = 0
    while(addOffspring < cutNum):
     parent1 = population[random.randint(0,(popSize-cutNum))]
     parent2 = population[random.randint(0,(popSize-cutNum))]
     while(parent1 == parent2):
             parent2 = population[random.randint(0,popSize-cutNum)]
     crossovered = crossover(parent1,parent2)
     mutated = mutation(crossovered,MUTATION_RATE)
     cutPopSet.append(mutated)
     addOffspring += 1
```

```
47      population = cutPopSet
48      generationCount += 1
49      return printSet
```

Ex.4: Parameters & Results

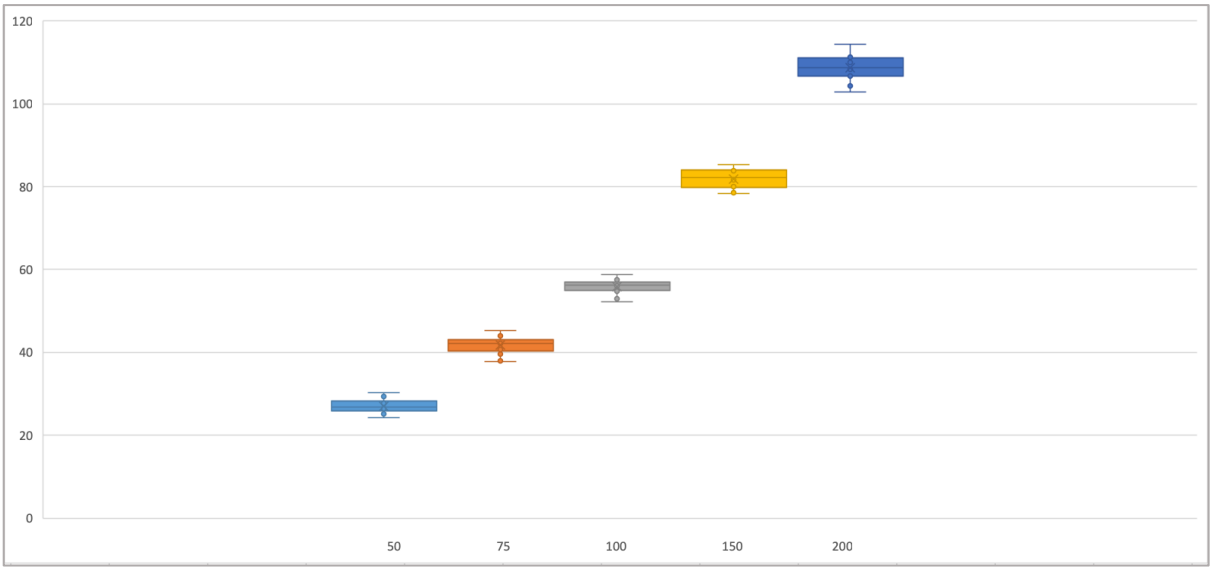
I will discuss and explain my parameter settings and results in this part. For each parameter setting, I did a 100 repetitions, and plotted box plots of the average attendance over several weeks.

Population Size

For this project, I used a preset value of crowded rate (60% of total population) to test the impact of different population sizes on results. The result is: the population size will not have a significant impact in my algorithm, because the crowded rate is based on the population size directly. Therefore I just used 100 as the population size for other tests.

Population Size = [50, 75, 100, 150, 200]
h(number of states) = 5; week number = 10;
generation = 100; crowded rate = 0.6; mutation rate = 0.1

Population Size	Expected Average Value (pop size * crowded rate)	Actual Average Value (100 repetitions)
50	30	27.068
75	45	41.791
100	60	55.836
150	90	81.914
200	120	108.705

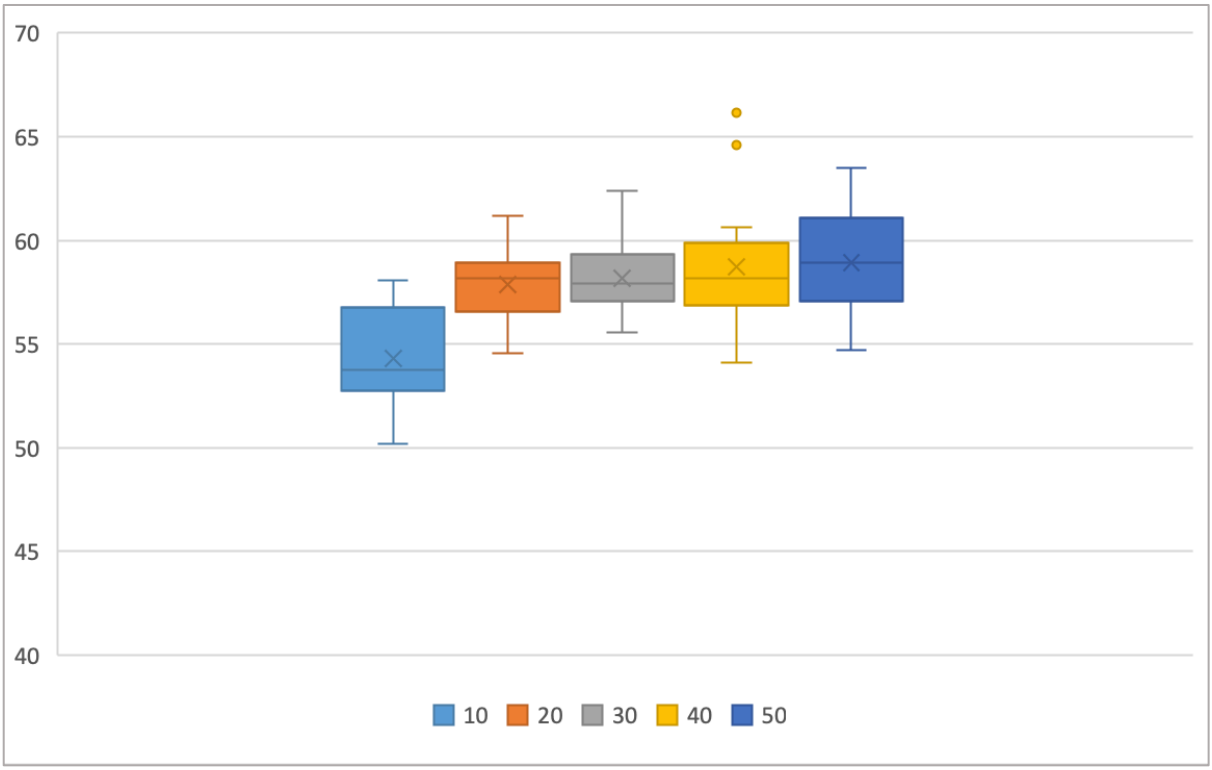


Weeks

In this section, I tested different week numbers to see the impact. I found that the larger week numbers, the closer result to expected value. However, if the setting of week number is too large, it will become very time consuming; and the standard deviation become large again. Therefore, I selected the week number as 30 for other tests.

week number = [10, 20, 30, 40, 50]
h(number of states) = 5; Population Size = 100;
generation = 100; crowded rate = 0.6; mutation rate = 0.1

Number of Weeks (per generation)	Expected Average Value (pop size*crowded rate)	Actual Average Value (100 repetitions)	Standard Deviation (100 repetitions)
10	60	54.286	2.265
20	60	57.852	1.754
30	60	58.174	1.597
40	60	58.726	2.771
50	60	58.914	2.489

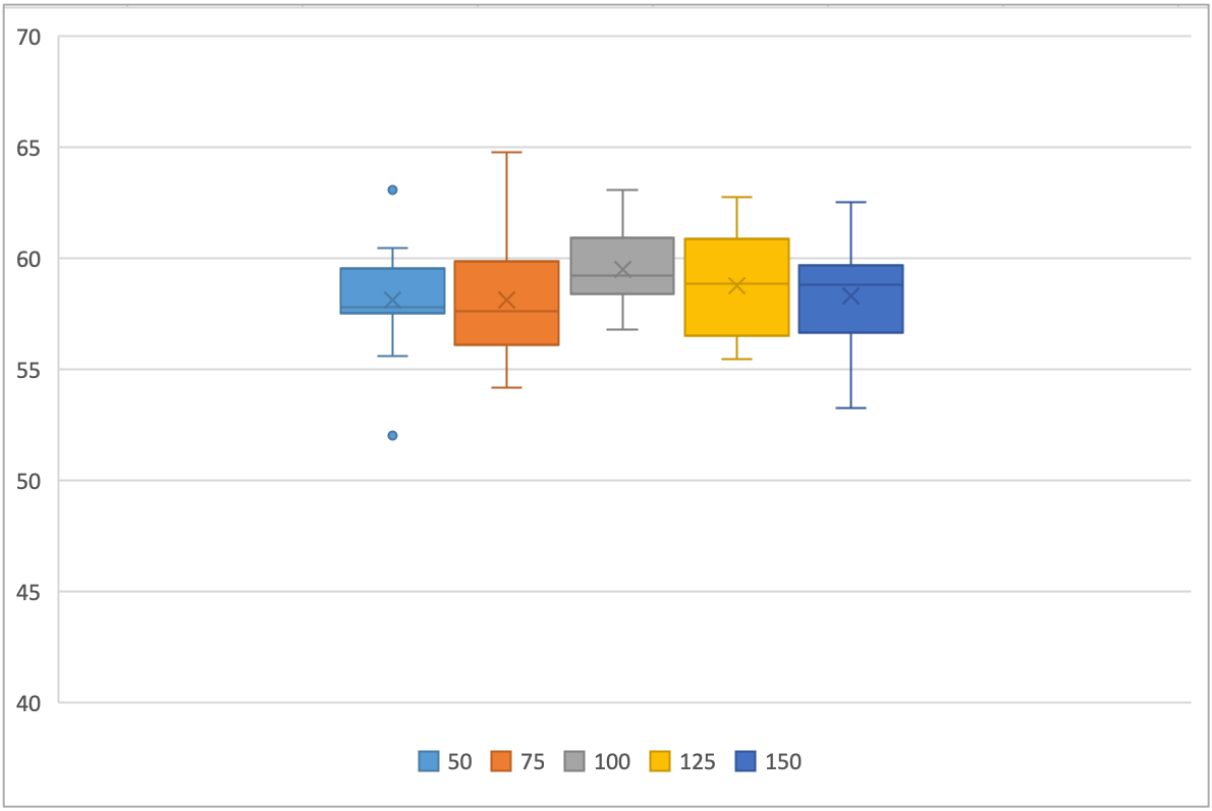


Generation

In this section, I tested five different generation settings to see the impact on results. The best generation setting in my algorithm is 100. If I set a number larger than 100, the standard deviation will become larger again. I deduced the reason is mutation and crossover settings making new population worse if it already reached the best population set (i.e. 100 times in my algorithm). Therefore, I selected 100 as my generation numbers for other tests.

generation number = [50, 75, 100, 125, 150]
h(number of states) = 5; Population Size = 100;
week number = 30; crowded rate = 0.6; mutation rate = 0.1

Number of Generation (per generation)	Expected Average Value (pop size*crowded rate)	Actual Average Value (100 repetitions)	Standard Deviation (100 repetitions)
50	60	58.124	2.256
75	60	58.134	2.708
100	60	59.503	1.781
125	60	58.758	2.204
150	60	58.311	2.203

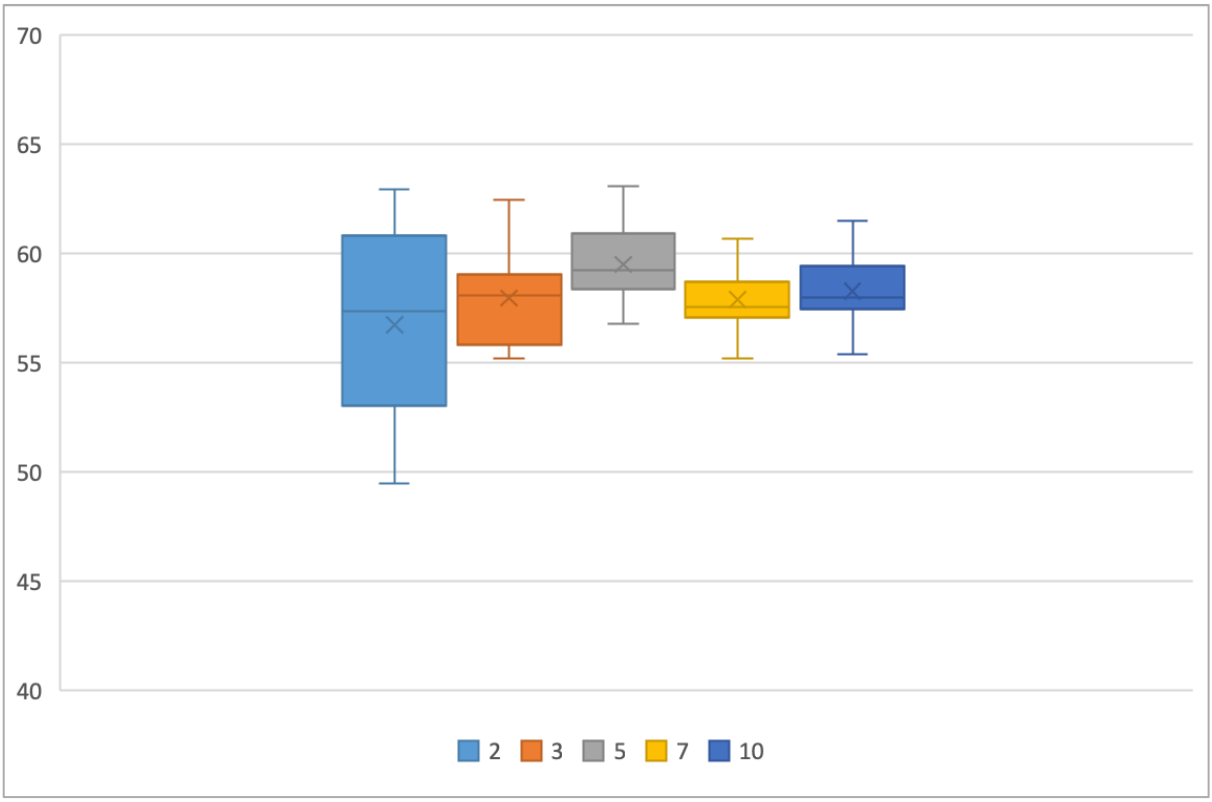


Number of States (h)

In this section, I tested different number of states to see the impact on results. The best result of setting is 7. However, the setting of 7 was very time consuming in my tests, but the setting of 5 was acceptable. The calculation time of 5 states is much smaller than 7 states. Therefore, I selected a mid-number as 5 for left tests.

h(number of states) = [2, 3, 5, 7, 10]
generation number = 100; Population Size = 100;
week number = 30; crowded rate = 0.6; mutation rate = 0.1

Number of States(h) (per generation)	Expected Average Value (pop size*crowded rate)	Actual Average Value (100 repetitions)	Standard Deviation (100 repetitions)
2	60	56.752	4.154
3	60	57.961	2.083
5	60	59.503	1.781
7	60	57.879	1.347
10	60	58.262	1.446

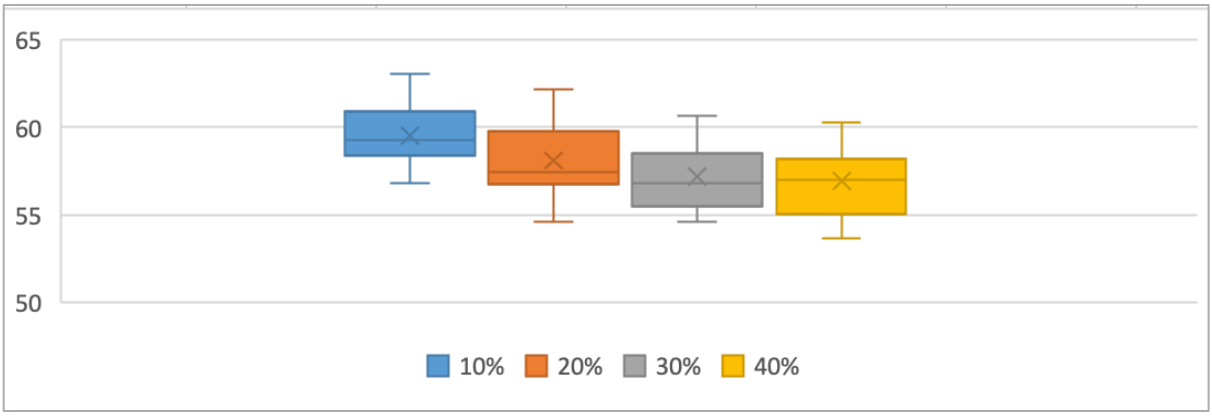


Mutation Rate

The mutation rate in my algorithm is actually the number of mutation in strategy. For example, if set mutation rate as 10%, the number of mutation should be ceil(0.1*5(states)*2) = 1. Each mutation operation will mutate either part A or part B in one state, therefore there is x2 in calculation. I tested different mutation rates in the setting of 5 states (h number), therefore the result for each rate (10%, 20%, 30% & 40%) should be like mutate (once, twice, 3 times & 4 times) for each generation.

mutation rate = [10%, 20%, 30%, 40%]
h(number of states) = 5; crowded rate = 0.6
generation number = 100; Population Size = 100; week number = 30

Mutation Rate (per generation)	Expected Average Value (pop size*crowded rate)	Actual Average Value (100 repetitions)	Standard Deviation (100 repetitions)
0.1 (once)	60	59.503	1.781
0.2 (twice)	60	58.103	2.013
0.3 (3 times)	60	57.174	1.893
0.4 (4 times)	60	56.941	1.907



Conclusion

Since the aim of parameter exam is to conduct the population to as efficient utilization of the bar as possible (i.e., close to 60%), while still not being overcrowded. The selected parameter due to mentioned reasons were [population size = 100, week number = 30, generation number = 100, h = 5 and mutation rate = 0.2].