PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

Diplomatura de Especialización en Desarrollo de Aplicaciones con Inteligencia Artificial

Optimización Industrial con Computación Evolutiva

Elaborado por:

FIGUEROA CARLOS, BRAD AXL

TORIBIO OSSIO, GERSON RICARDO

2019-1

- **Problemática**

El problema consiste en que una conocida cadena de supermercados desea ingresar al mercado limeño, para eso se posee 60 locales candidatos donde se podría localizar los supermercados. Por lo que solo se desea colocar 10 posibles locales.

- **Solución mono-objetivo:**

Se utilizará una clase llamada **Market**, esta clase tendrá los siguientes atributos:

```
class Market(object):
    def __init__(self,ID,descr,pob,latitud,longitud):
        self.id=ID
        self.descr=descr
        self.pob=pob
        self.lat=latitud
        self.lon=longitud
```

Esta será la estructura que usaremos para la representación de los individuos, donde cada individuo tiene la información importante brindada el *archivo .csv.*

En este caso se utilizó el *cruzamiento de manera uniforme* debido a que brinda variedad y no escoge los mejores cromosomas creando así una elite. Cabe recalcar que nuestra función posee una llamada a una función que verifica si al hacer el cruzamiento los cromosomas resultantes poseen la cantidad de valores 1 igual a 10. Esta función se llama *__verify,* si en caso nuestro cromosoma posee más de 10 alelos con valor 1 entonces se quitará los alelos 1 hasta que posea 11 alelos con valor 1 dentro del cromosoma. Caso contrario, si nuestro cromosoma posee menos de 10 alelos con valor 1 entonces se generará alelos aleatorios para completar la falta de alelos con valor 1.

Además, se utilizará la función de mutación *mutation_flip,* esta función cambia el alelo de un gen. Sin embargo, este algoritmo posee una variación con respecto al visto en clase, ya que cada vez que se agregue un uno, deberá verificarse que ese valor no agregue alelos 1 de más haciendo que nuestro cromosoma tenga un alelo adicional con el valor de 1 y de esta forma nos arroje 11 valores con alelo igual a 1 y no sea una solución viable para nuestro problema.

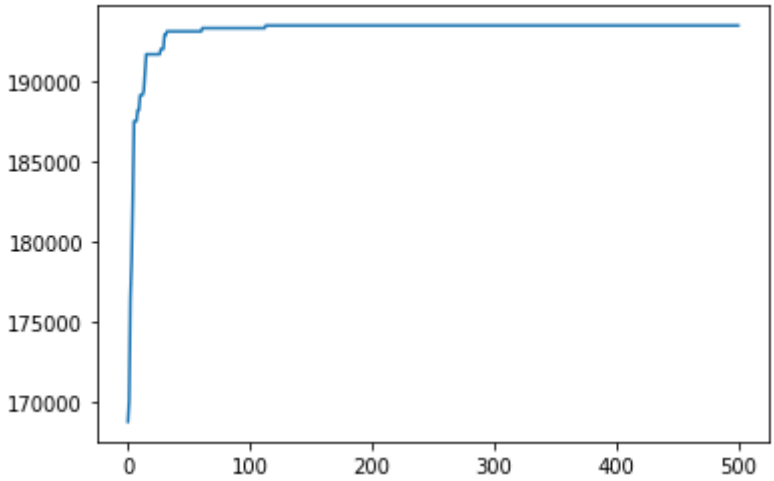
Para escoger los individuos se utilizó el método de *selección por ruleta*.

Luego nuestra función fitness retorna el cálculo de las distancias entre los locales seleccionados. Para este punto, se calculó previamente las distancias de un posible local con todos los otros posibles locales. Esto ayuda a que no se esté realizando el cálculo entre locales debido a que posee una distancia fija.

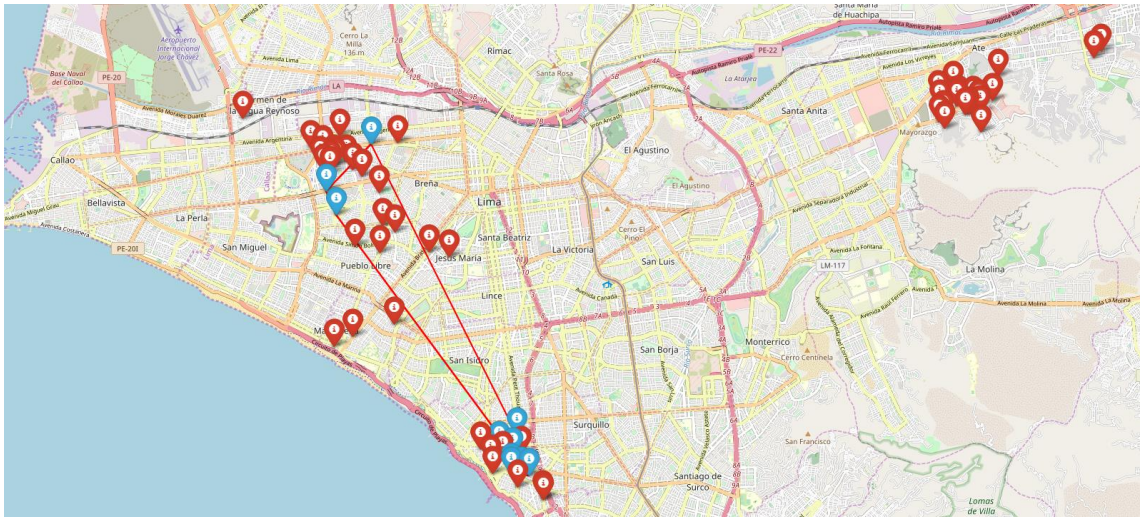
1. *Ejecución del algoritmo:*
   Para la ejecución del algoritmo genético se utilizó los siguientes hiperparámetros

```
## Hiperparametros del algoritmo genetico
POP_SIZE = 50       # numero de individuos
GENERATIONS = 500   # numero de generaciones
PMUT = 1        # tasa de mutacion

## Inicializa una poblacion inicial de forma aleatoria
population = init_population(POP_SIZE,maxMark=10)

# Evolue la poblacion con el algoritmo genetico (cruzamiento 'onepoint', )
best_ind, bestfitness, history_population = genetic_algorithm(population, marketPool, GENERATIONS, PMUT, crossover="uniform", mutation="flip",select='roulette')
```

Y como resultado se obtuvo la siguiente gráfica donde se indica en que generación (Eje x) nuestro algoritmo obtuvo la solución óptima para el algoritmo mono-objetivo.



Se observa que casi en la generación 150 se queda de manera estable el crecimiento del fitness. Este fitness fue de **193409.90950866626**. Y la suma de las distancias fue de **10309.909509** la cual sería una de las máximas distancias entro los posibles locales. Analizando de manera gráfica se puede obtener los siguiente:



En donde los puntos azules indican los puntos que se escogieron como solución para nuestro algoritmo genético.

2. *Evaluación del algoritmo genético de manera iterativa:*

Este algoritmo fue ejecutado 10 veces con los parámetros anteriormente mencionados. Este algoritmo arrojo el mismo resultado promedio de fitness y suma de distancias.

	individuos	fitness	suma_distancia	suma_poblacion
0	<__main__.Individual object at 0x7fa281111b70>	193409.909509	10309.909509	183100
1	<__main__.Individual object at 0x7fa28109c7f0>	193409.909509	10309.909509	183100
2	<__main__.Individual object at 0x7fa2814d8940>	193409.909509	10309.909509	183100
3	<__main__.Individual object at 0x7fa28109c128>	193409.909509	10309.909509	183100
4	<__main__.Individual object at 0x7fa281079f98>	193409.909509	10309.909509	183100
5	<__main__.Individual object at 0x7fa28104c438>	193409.909509	10309.909509	183100
6	<__main__.Individual object at 0x7fa27dd271d0>	193409.909509	10309.909509	183100
7	<__main__.Individual object at 0x7fa280bc6b38>	193409.909509	10309.909509	183100
8	<__main__.Individual object at 0x7fa27dd270f0>	193409.909509	10309.909509	183100
9	<__main__.Individual object at 0x7fa28104ce80>	193409.909509	10309.909509	183100

El resultado obtenido fue el siguiente, donde todos arrojaron el mismo fitness.

- **Solución multi-objetivo:**

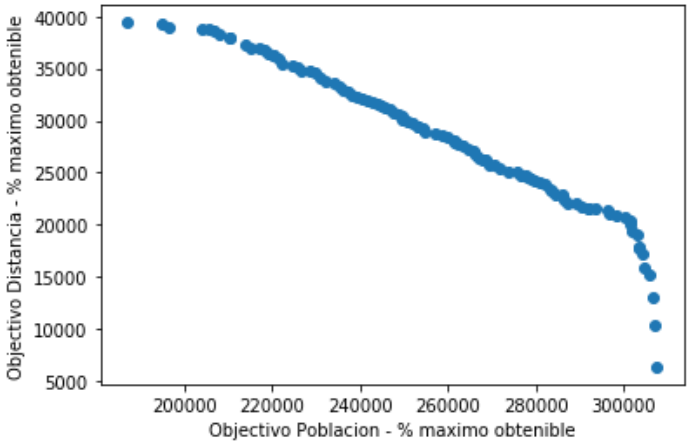
  Para la solución multiobjetivo se utilizó la misma estructura planteada en el punto anterior utilizando la clase **Market**. La diferencia con algoritmo anterior es que se utilizó el concepto de la frontera de Pareto y para esta ocasión se evaluará los centros comerciales más lejanos entre ellos y los que arrojen mayor población a los locales, lo cual indica que es una solución que posee más de un objetivo por maximizar.

  Este fue el algoritmo que se utilizó para generar la frontera de Pareto:

```
## Hiperparametros del algoritmo genetico
POP_SIZE = 50       # numero de individuos
GENERATIONS = 500   # numero de generaciones
PMUT = 1        # taza de mutacion
MIN_POP_SIZE = 100    # numero de individuos minimo
MAX_POP_SIZE = 100    # numero de individuos maximo

## Inicializa una poblacion inicial de forma aleatoria
P = init_population(POP_SIZE,maxMark=10)

evaluate_population_nsga(P,marketPool)    #  evalua la poblacion
## Ejecuta los ciclos evolutivos
for g in range(GENERATIONS):   # Por cada generacion

    if g %10 == 0:
        print ('Generacion {} (de {}) '.format(g, GENERATIONS))

    ## genera y evalua la poblacion hija
    Q = build_offspring_population(P, "uniform", "flip", PMUT)
    evaluate_population_nsga(Q, marketPool)

    ## une la poblacion padre y la poblacion hija
    P.extend(Q)

    ## Construye la poblacion de la siguiente generacion
    P = build_next_population(P, MIN_POP_SIZE, MAX_POP_SIZE)

# Obtiene la poblacion de la frontera de pareto final
pareto_front_population = get_paretofront_population(P)
```

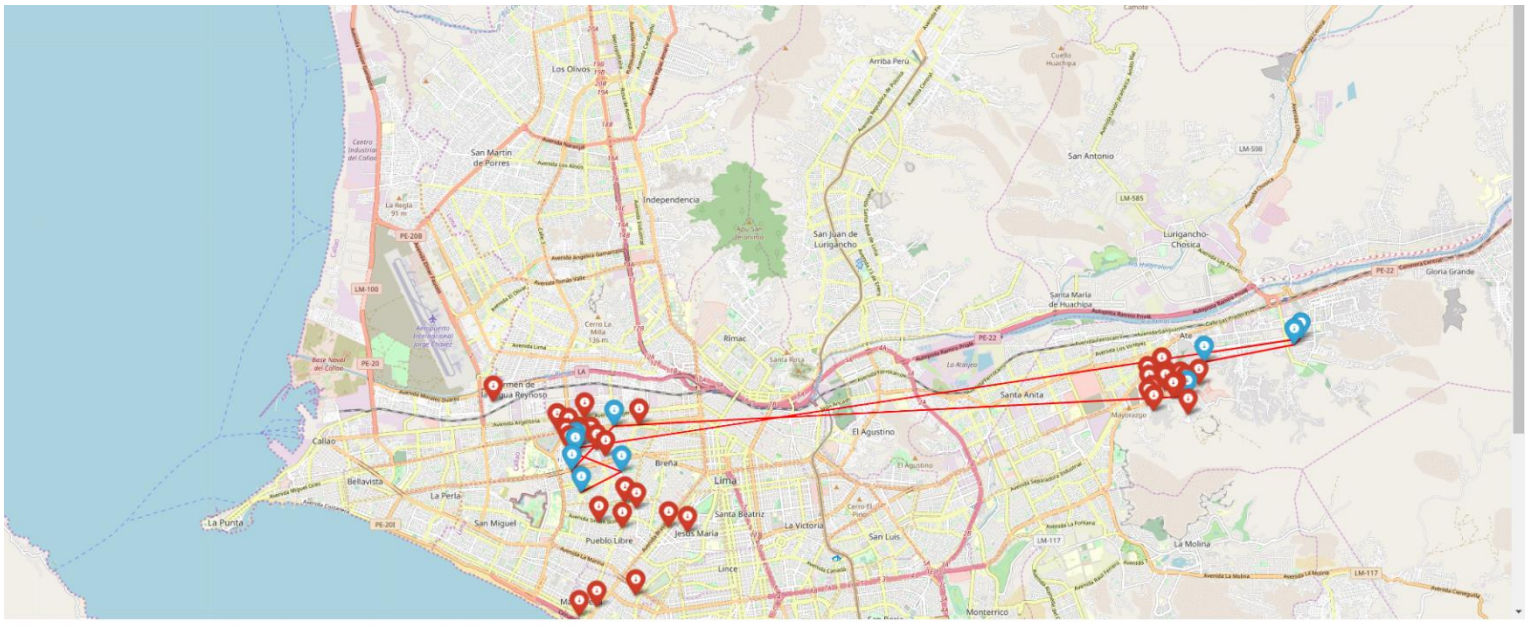
Donde los hiperparámetros están en la parte superior del código.

Se obtuvo el siguiente resultado de la frontera de Pareto:



En este caso se analizó de manera geográfica las soluciones de Pareto en los límites. En otras palabras, cuando se maximiza la distancia, cuando se maximiza la población o cuando ambas variables están maximizados.

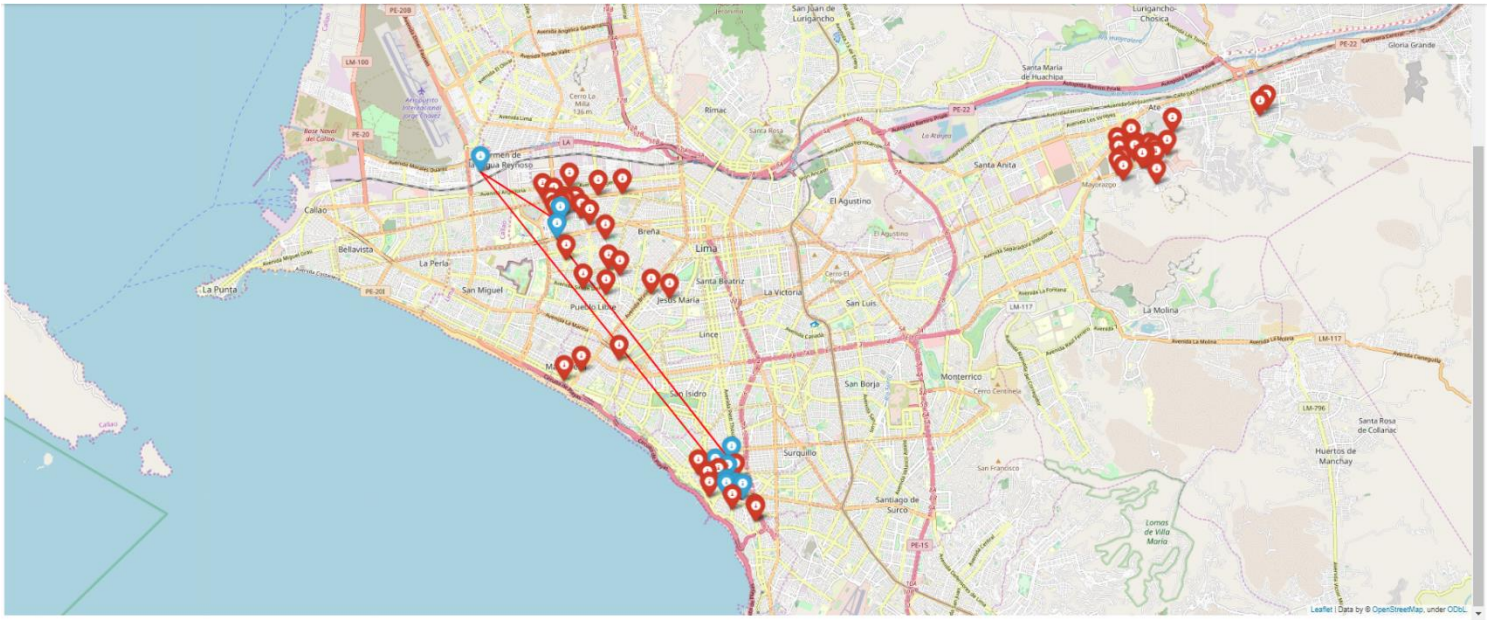
- Solución de la frontera de Pareto con maximización de las distancias:




```
1 print("Esta solución tiene como cantidad de población {} y un maximo de distancia de entre locales de {}".format(individuo_dist.fitness[0]/CHROMOSOME_SIZE*100, individuo_dist.fitness[1]/CHROMOSOME_SIZE*100))
```

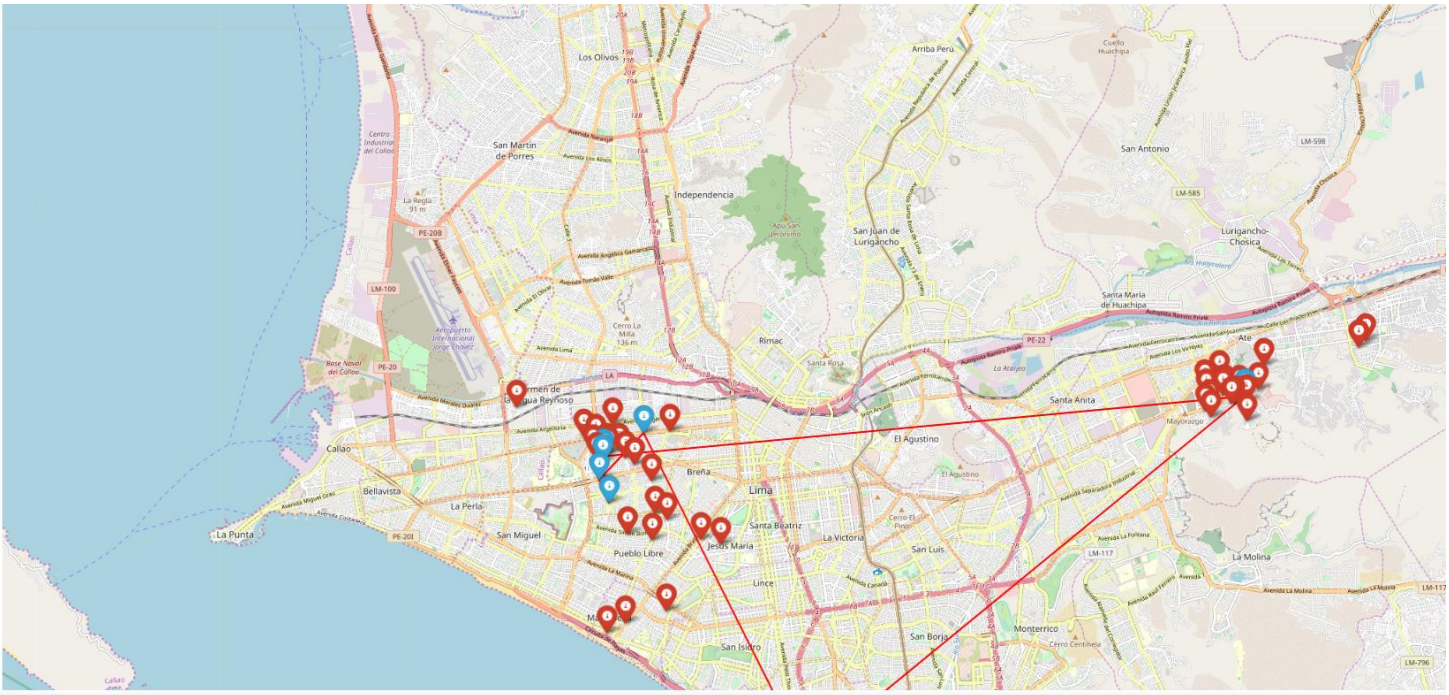
```
Esta solución tiene como cantidad de población 193083.3333333333 y un maximo de distancia de entre locales de 39241.73627165429
```

- Solución de Pareto con maximización de la población:




```
1 print("Esta solución tiene como maximo fitness cantidad de población {} y una distancia fitness entre locales de {}".format(individuo_pob.fitness[0]/CHROMOSOME_SIZE*100, individuo_pob.fitness[1]/CHROMOSOME_SIZE*100))
```

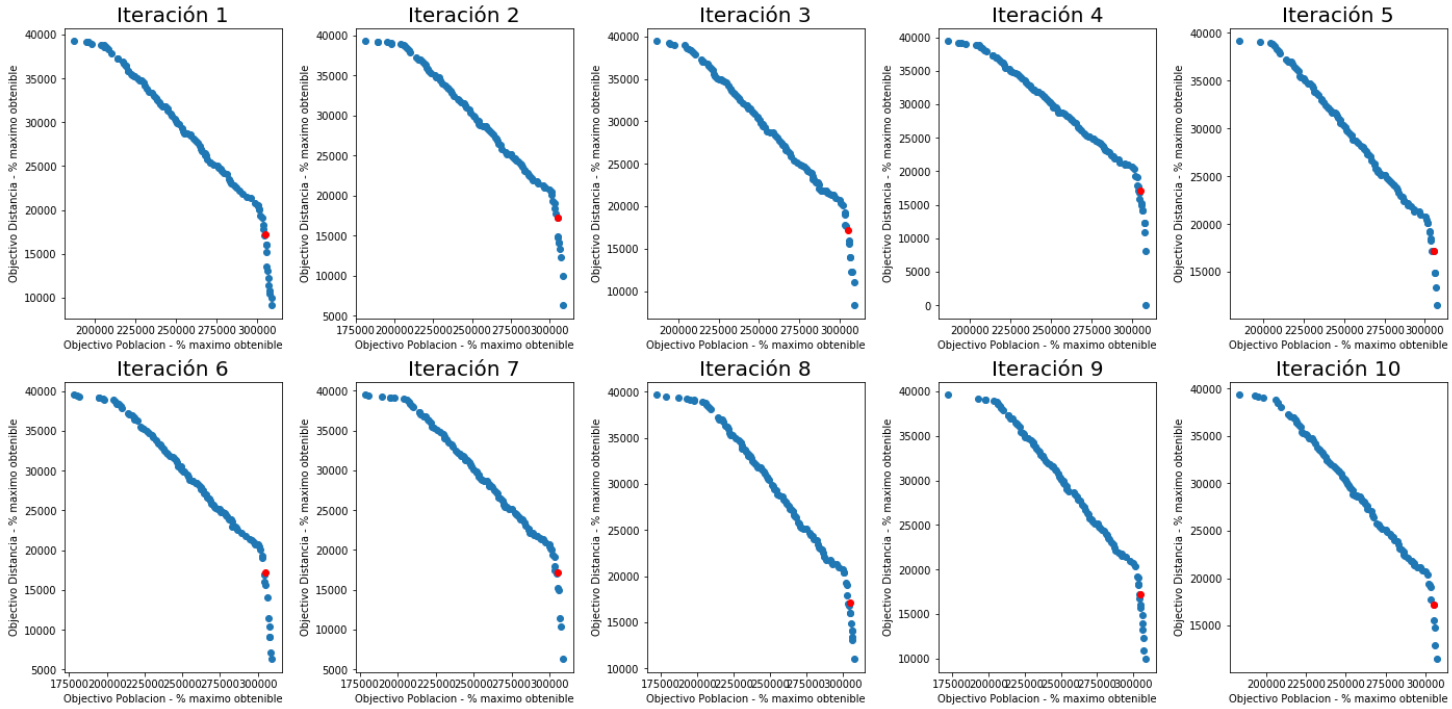
- Solución de Pareto con maximización de ambos parámetros




```
1 print("Esta solución tiene como cantidad de población {} y la distancia de entre locales de {}".format(individuo_medio.fitness[0]/CHROMOSOME_SIZE*100, individuo_medio.fitness[1]/CHROMOSOME_SIZE*100))
```

```
Esta solución tiene como cantidad de población 283750.0 y la distancia de entre locales de 23183.530369021522
```

Nuestro Algoritmo genético multi-objetivo fue ejecuta 10 veces con los hiperparametros mencionados anteriormente. Por lo que se obtuvo el siguiente gráfico para cada iteración.



- **Comparación de los algoritmos multi-objetivo y mono-objectivo**

  En conclusión, se obtuvo la frontera de Pareto para el algoritmo multi-objetivo y la solución óptima para el mono-objetivo. Se notó que la distancia entre los locales tiende es afectada por la población y esto se debe a que la escala de los valores entre las distancias y la cantidad de población. La diferencia entre estos dos valores es demasiado grande por lo cual el cálculo del fitness se ve afectado y tiende siempre a obtener los locales que contengan mayor población. Por lo que se llega a la conclusión que la solución de la frontera de Pareto da mejores resultados en comparación con la mono-objetivo la cual se deja influenciar demasiado por la diferencia de escala entre las variables a optimizar