Grupo·epm

Optimización de ruteos de Actividades de Mantenimiento mediante algoritmos genéticos

Sergio Quintero

Problema a resolver

Largos recorridos en el mantenimiento

Debido a la gran cantidad de activos que se tiene en el sistema de acueducto y alcantarillado, a los se debe de realizar mantenimiento periódico para garantizar su adecuada funcionalidad, el objetivo es encontrar la forma óptima en que se pueden visitar cada uno de estos elementos teniendo en cuenta las distancias entre estos.

Objetivo

Encontrar la distancia minima de recorridos en Mantenimiento

Optimización

La optimización es la acción de desarrollar una actividad lo más eficientemente posible, es decir, con la menor cantidad de recursos y en el menor tiempo posible.

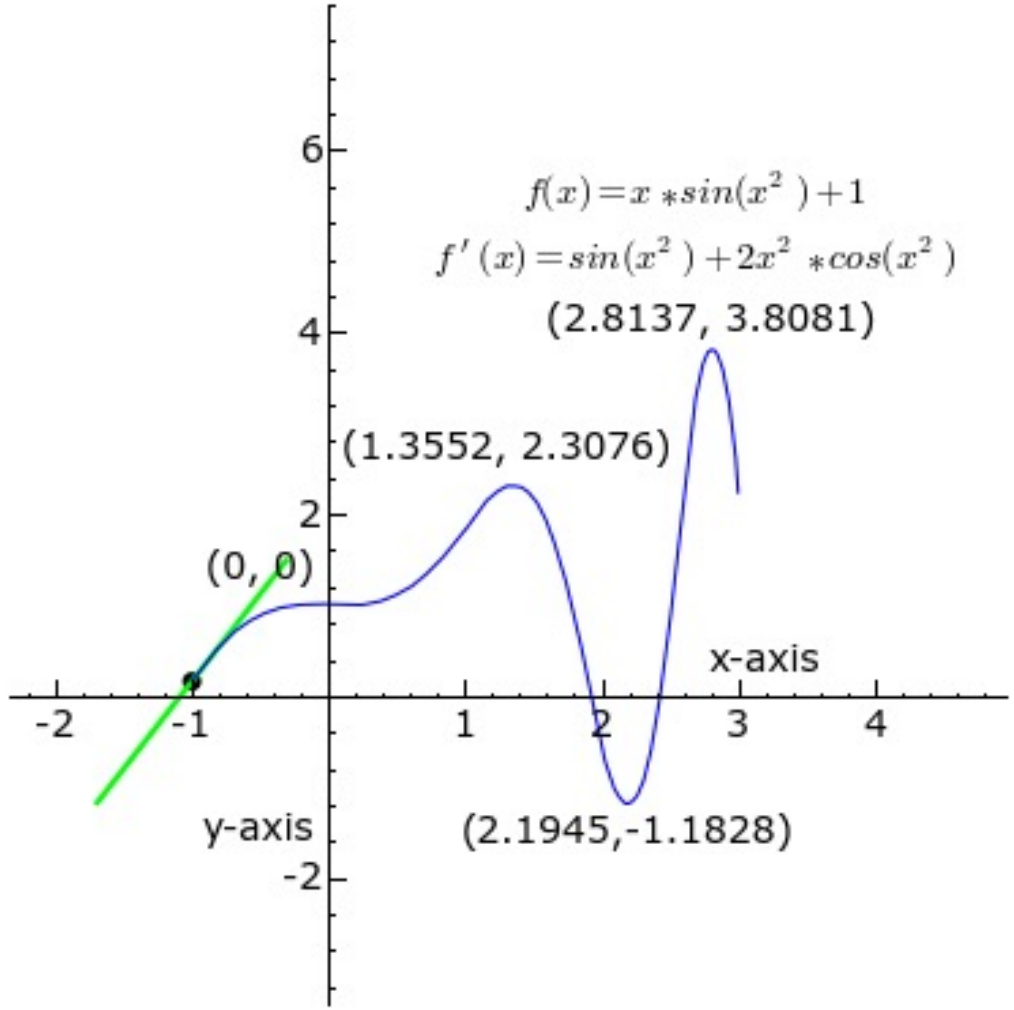
La optimización, en general, implica lograr el mejor funcionamiento de algo, usando de la mejor forma los recursos (www.economipedia.com)

Solución óptima

¿Cómo resolver un problema de optimización?

- Fuerza bruta
- Búsqueda aleatoria
- Técnicas basadas en gradients
- Algoritmos heurísticos

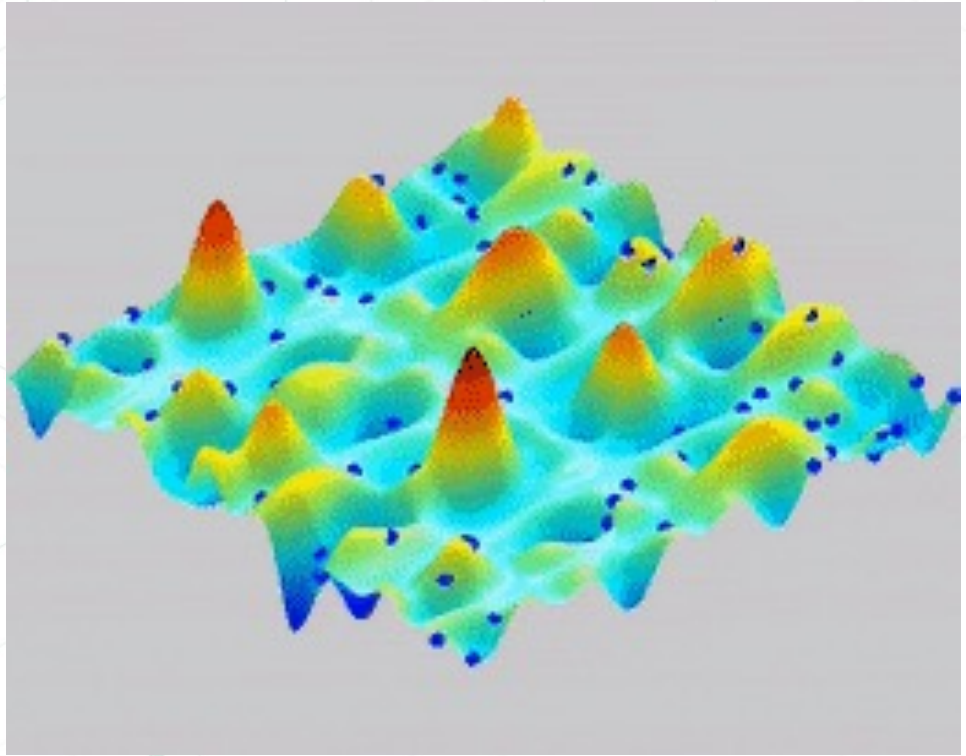
Problemas P y NP

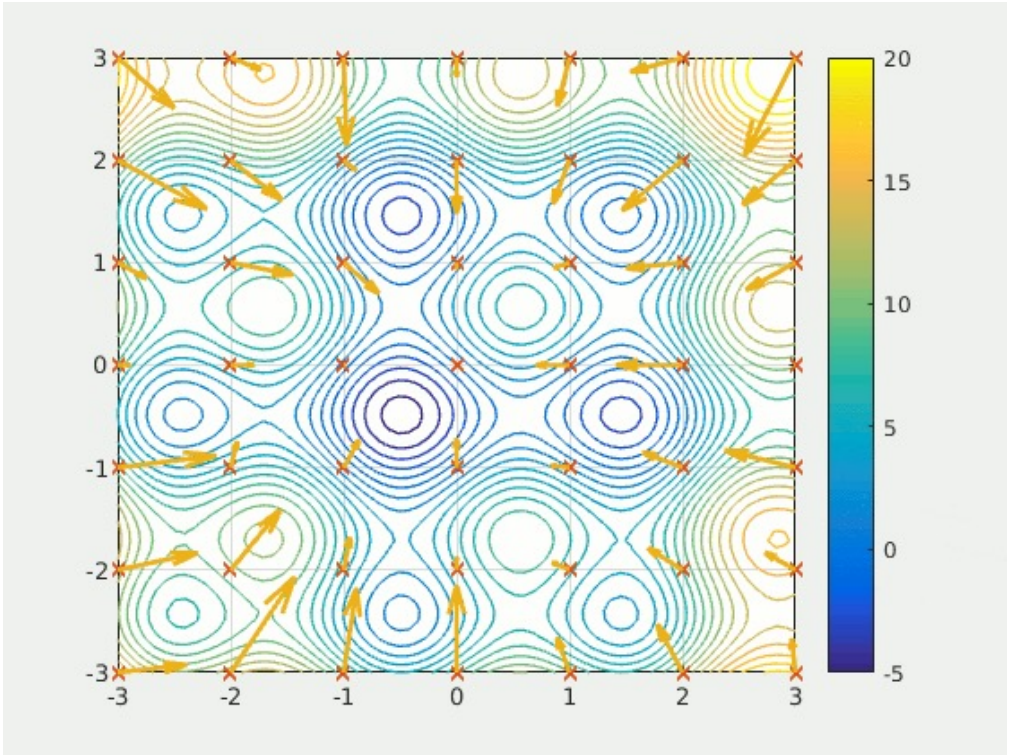


https://www.claymath.org/millennium-problems

Problemas complejos

Corresponde a problemas que debido a su complejidad o gran cantidad de iteraciones no es viable resolver mediante técnicas exactas de optimización





Algoritmos Heurísticos

Algoritmos que buscan dar soluciones a problemas muy difíciles mediante la simplificación del mismos o toma de atajos

Algoritmos Metaheurísticos

Son algoritmos heurísticos que buscan explorar un espacio de búsqueda de manera más eficiente que los algoritmos exactos

Analítica para la toma de decisiones

Machine Learning

Reinforcement Learning

Deep Learning

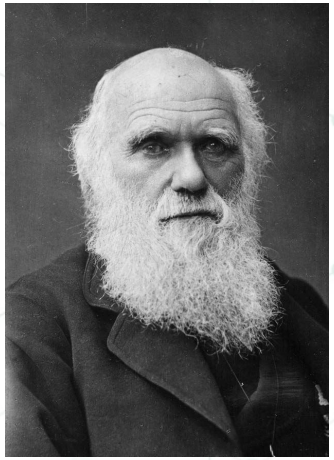
Algoritmos metaheuristicos

Inteligencia Artificial

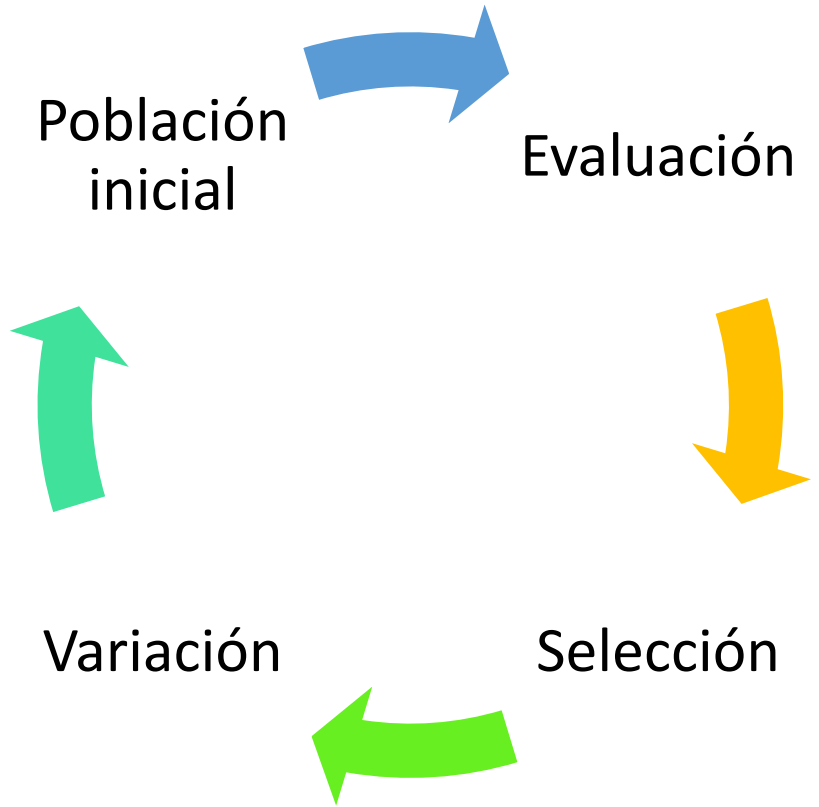
Algoritmos Genéticos

Computación evolutiva

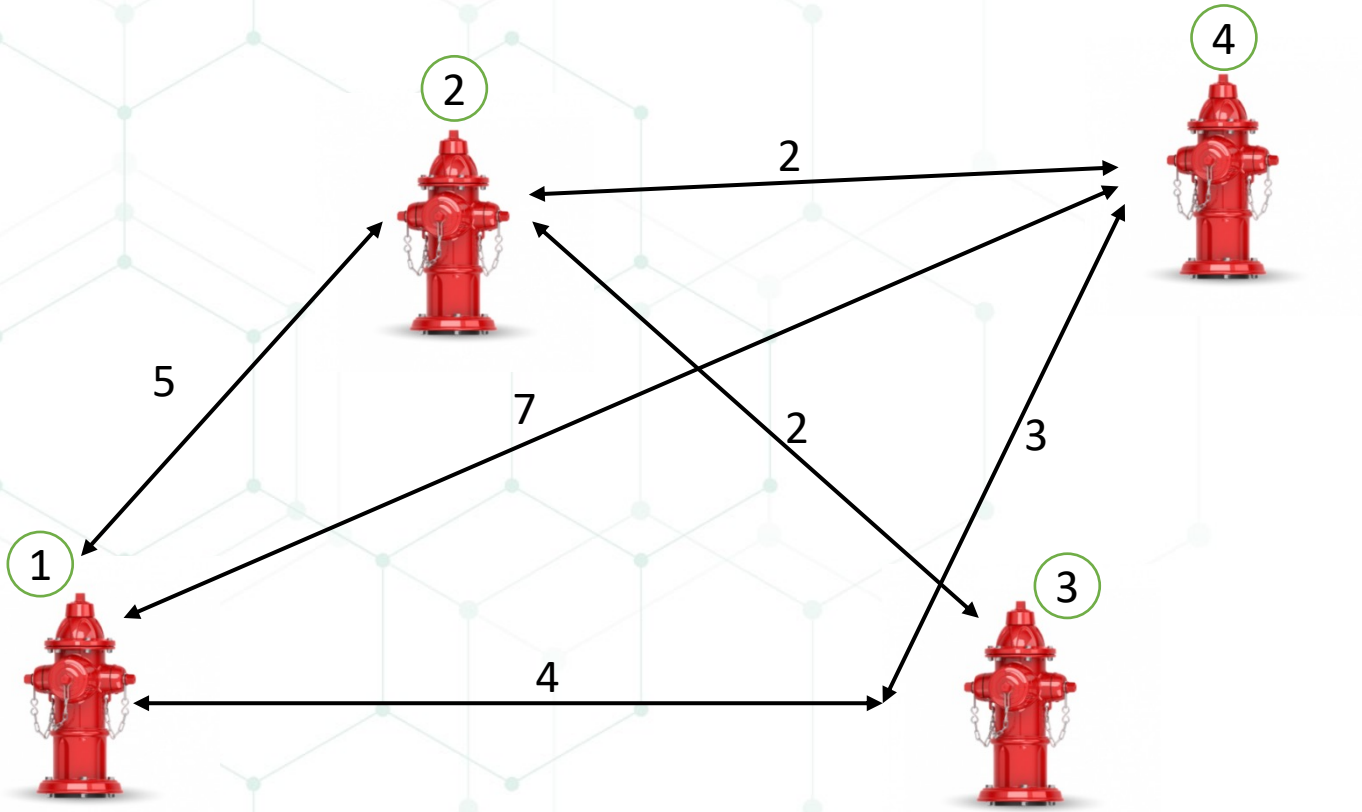
Los algoritmos genéticos buscan encontrar la solución a algunos problemas imitando los mecanismos que condicionan la evolución biológica



Los individuos que mejor se adaptan al medio son aquellos que tienen más probabilidades de dejar descendencia, y que por lo tanto, sus genes pasarán a las siguientes generaciones



Alternativas de solución



#Elementos (N)	N!/2
4	12
5	60
6	360
10	1.300.000 (7 dígitos)
50	65 dígitos
100	161 dígitos
300	623 dígitos

Modelo de optimización

Situación inicial

Hidrante 1	*Hidrante 2*	*Hidrante 3*	*Hidrante 4*	*Hidrante 5*
				

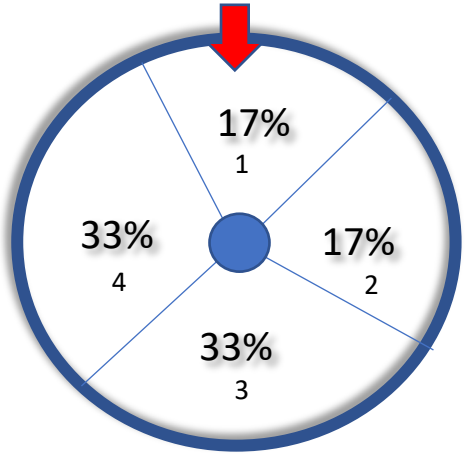
1 Generar población inicial

Ind1	1	4	5	3	2
Ind2	2	1	5	4	3
Ind3	1	2	5	3	4
Ind4	3	1	2	4	5

4	33%
3	33%
2	17%
1	17%

$$d(p1,p2) = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

2 Selección de individuos



Ind1	1	4	5	3	2
Ind2	2	1	5	4	3
Ind3	1	2	5	3	4
Ind4	3	1	2	4	5

3 Cruzamiento

Ind3	1	2	5	3	4
Ind4	3	1	2	4	5

Ind5	1	2	2	4	5

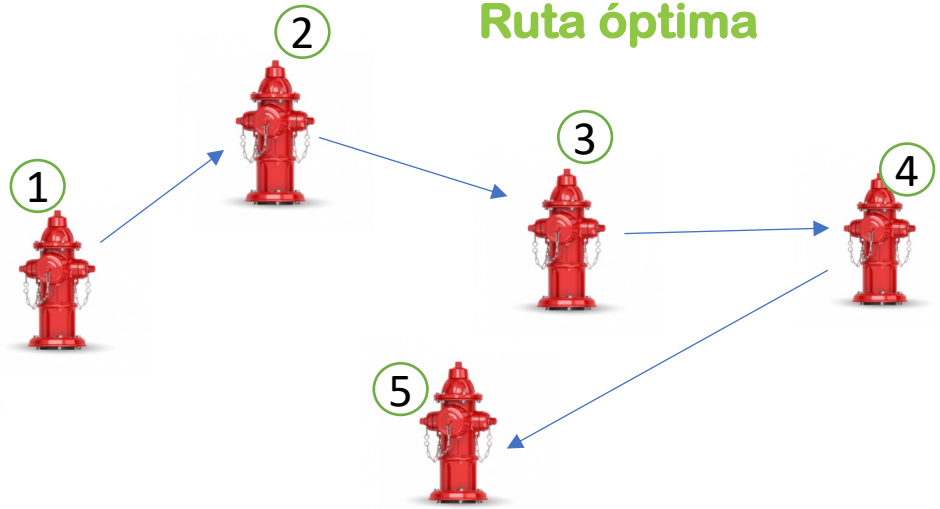
4 Mutación

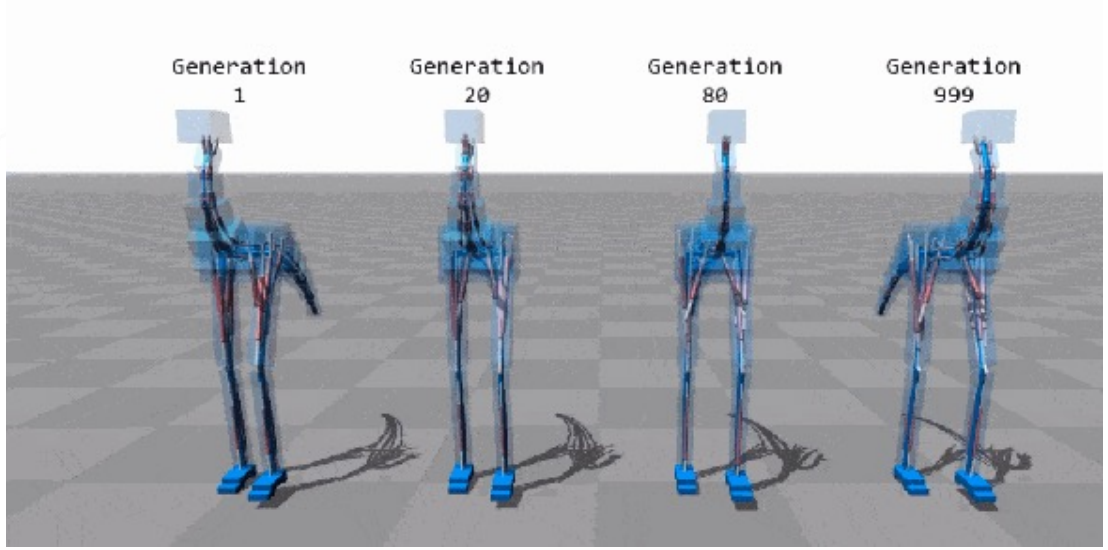
Ind5	1	2	3	4	5

Resultado final

Ind5	1	2	3	4	5	100%

Ruta óptima





Implementación modelo

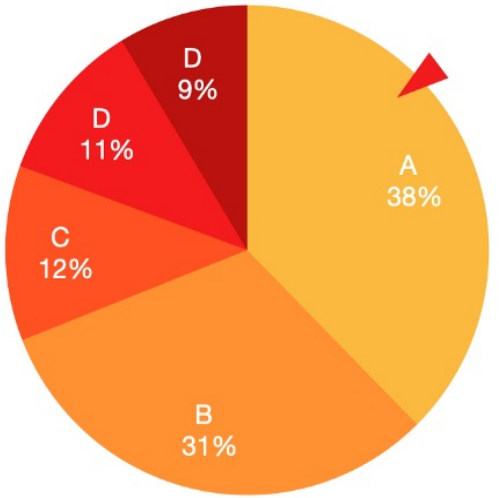
```python
class activo:
        def __init__(self,ipid,x,y):
            self.ipid = ipid
            self.x = x
            self.y = y

        def distance(self, elemento):
            xDis = abs(self.x - elemento.x)
            yDis = abs(self.y - elemento.y)
            distance = np.sqrt((xDis ** 2) + (yDis ** 2))
            return distance

        def __repr__(self):
            return "(" + str(self.ipid)+ ")"


        #devuelve el valor del ipid del activo
        def getname(self):
            return self.ipid

        #devuelve el valor de x del activo
        def getx(self):
            return self.x

        #devuelve el valor de y del activo
        def gety(self):
            return self.y
```

```python
class Fitness:
    def __init__(self, route):
        self.route = route
        self.distance = 0
        self.fitness= 0.0

    def routeDistance(self):
        if self.distance ==0:
            pathDistance = 0
            for i in range(0, len(self.route)):
                fromActive = self.route[i]
                toActive = None
                if i + 1 < len(self.route):
                    toActive = self.route[i + 1]
                else:
                    toActive = self.route[0]
                pathDistance += fromActive.distance(toActive)
            self.distance = pathDistance
        return self.distance

    def routeFitness(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.routeDistance())
        return self.fitness
```

```python
def createRoute(activeList):
    route = random.sample(activeList, len(activeList))
    return route
```

```python
def rankRoutes(population):
    fitnessResults = {}
    for i in range(0,len(population)):
        fitnessResults[i] = Fitness(population[i]).routeFitness()
    sorted_results=sorted(fitnessResults.items(), key = operator.itemgetter(1), reverse = True)
    return sorted_results
```

```python
def selection(popRanked, eliteSize):
    selectionResults = []
    df = pd.DataFrame(np.array(popRanked), columns=["Index","Fitness"])
    df['cum_sum'] = df.Fitness.cumsum()
    df['cum_perc'] = 100*df.cum_sum/df.Fitness.sum()

    for i in range(0, eliteSize):
        selectionResults.append(popRanked[i][0])
    for i in range(0, len(popRanked) - eliteSize):
        pick = 100*random.random()
        for i in range(0, len(popRanked)):
            if pick <= df.iat[i,3]:
                selectionResults.append(popRanked[i][0])
                break
    return selectionResults
```

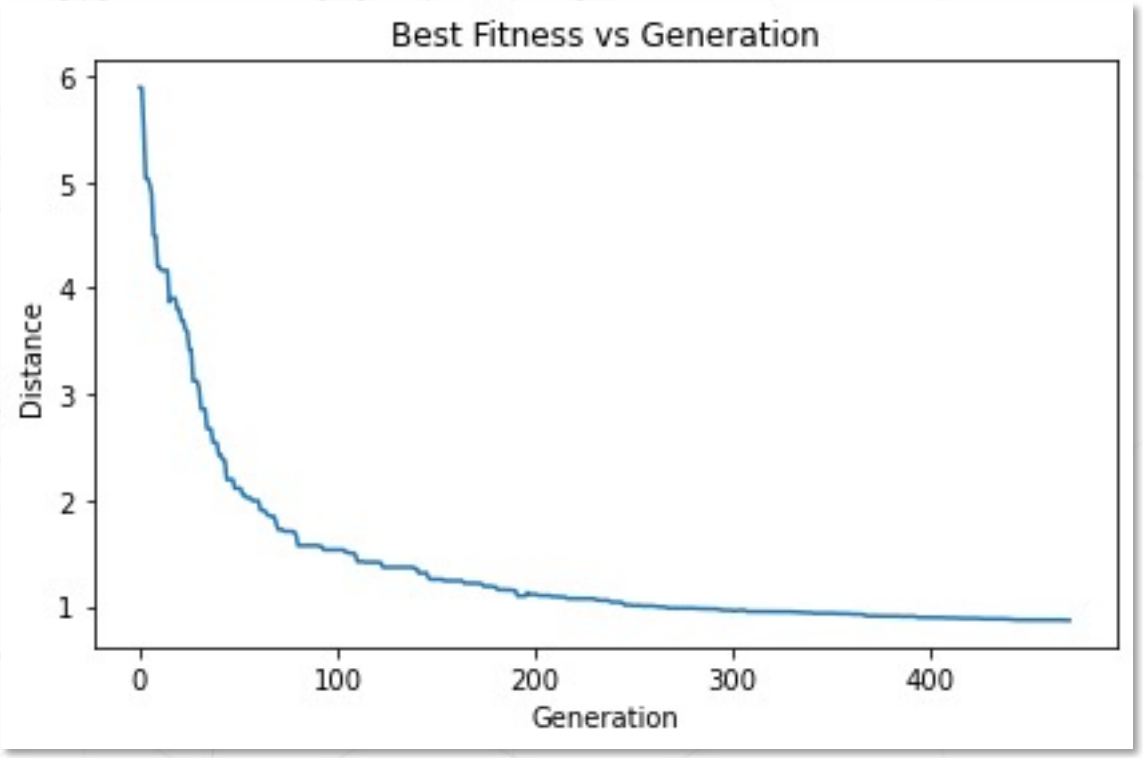


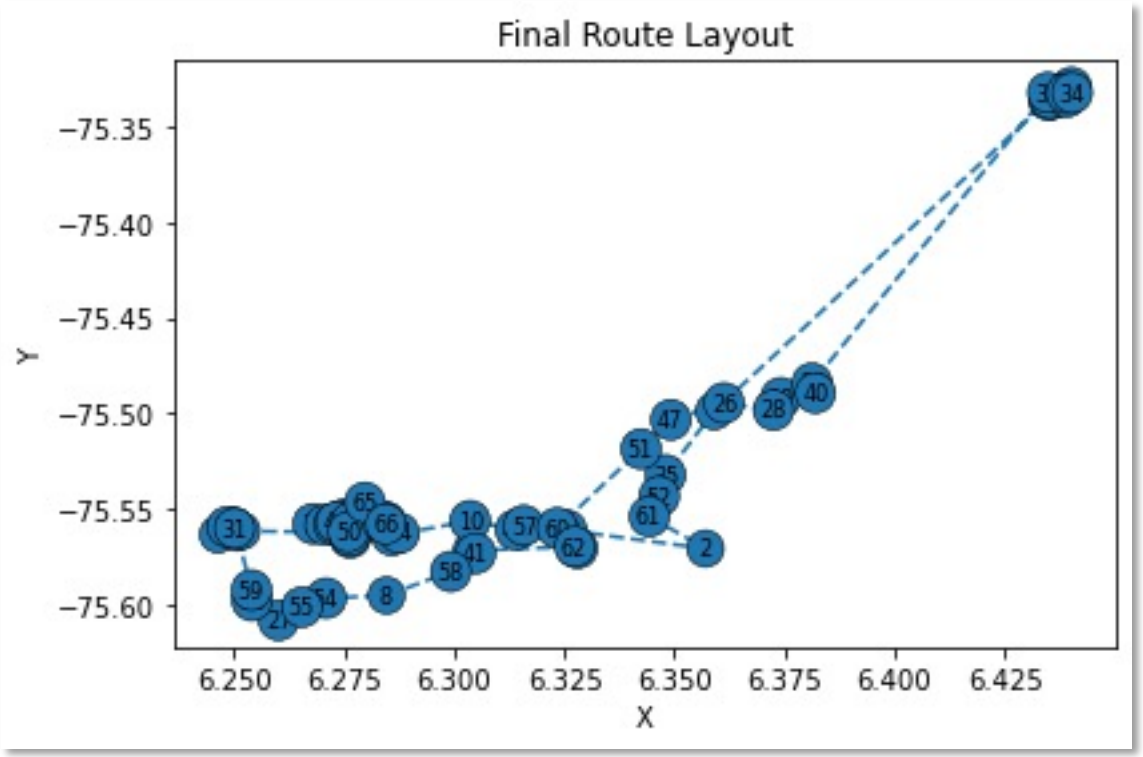
```python
def breed(parent1, parent2):
    child = []
    childP1 = []
    childP2 = []

    geneA = int(random.random() * len(parent1))
    geneB = int(random.random() * len(parent1))

    startGene = min(geneA, geneB)
    endGene = max(geneA, geneB)

    for i in range(startGene, endGene):
        childP1.append(parent1[i])


    childP2 = [item for item in parent2 if item not in childP1]
    print(startGene, endGene)

    print(parent1)
    print(parent2)

    print(childP1)
    print(childP2)
    child = childP1 + childP2

    print(child)
    return child
```

```
def mutate(individual, mutationRate):
    for swapped in range(len(individual)):
        if(random.random() < mutationRate):
            swapWith = int(random.random() * len(individual))

            active1 = individual[swapped]
            active2 = individual[swapWith]

            individual[swapped] = active2
            individual[swapWith] = active1
    return individual
```

Resultados obtenidos

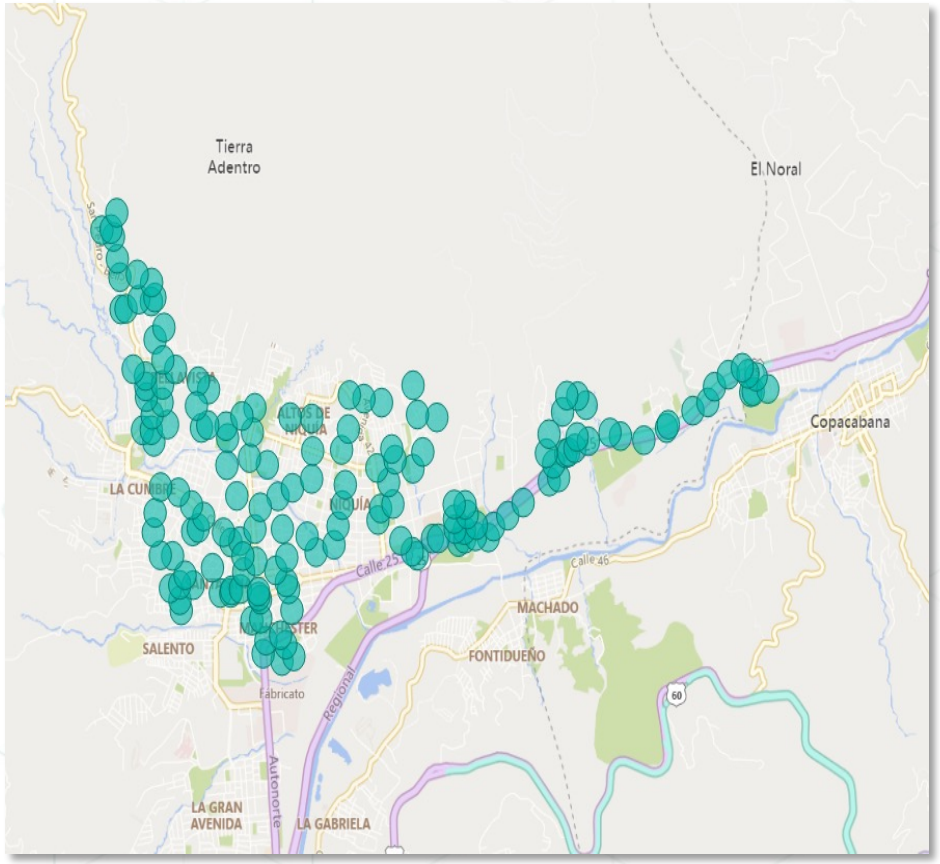
500 generaciones después…



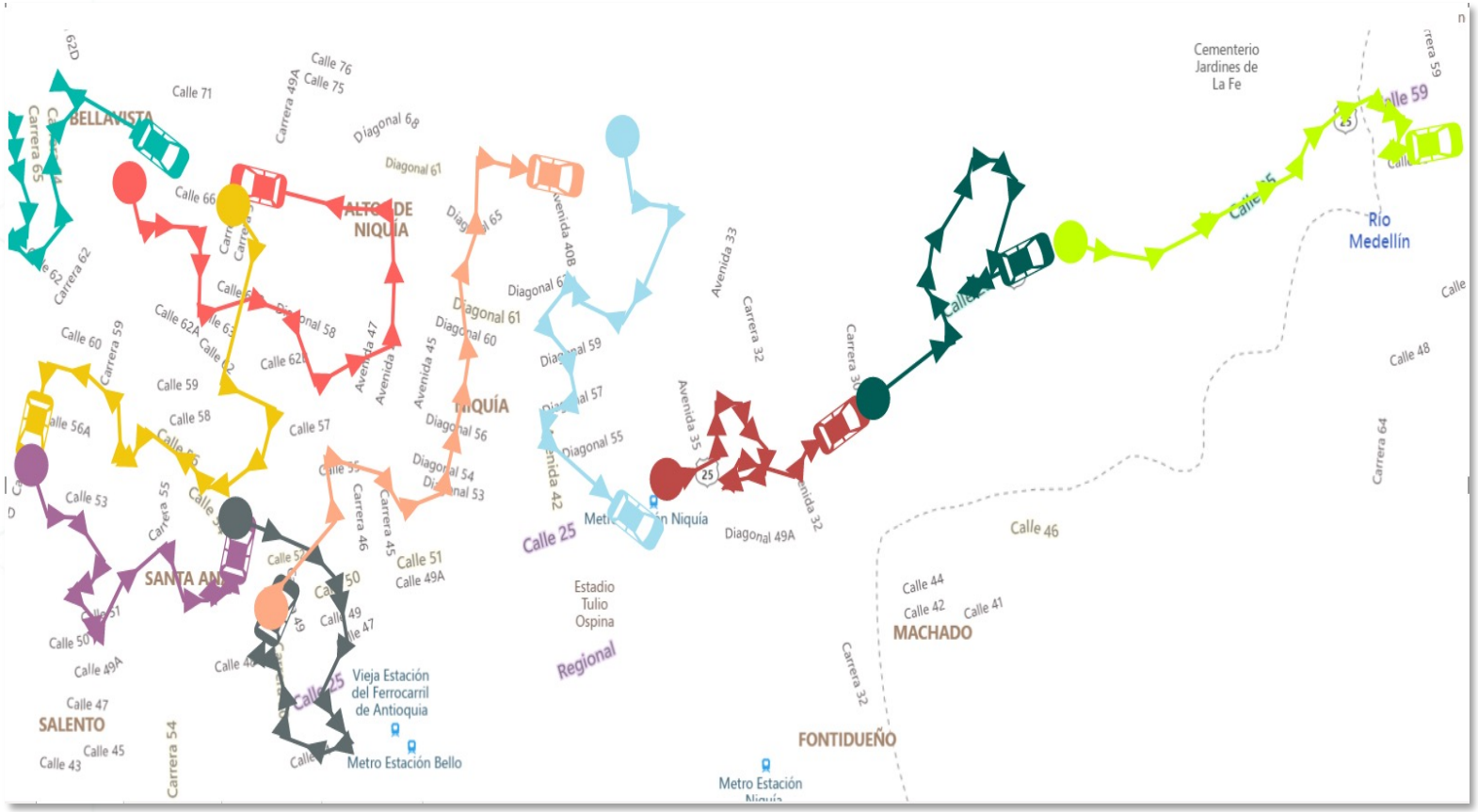


Resultados obtenidos

Problema a resolver



Solucón óptima

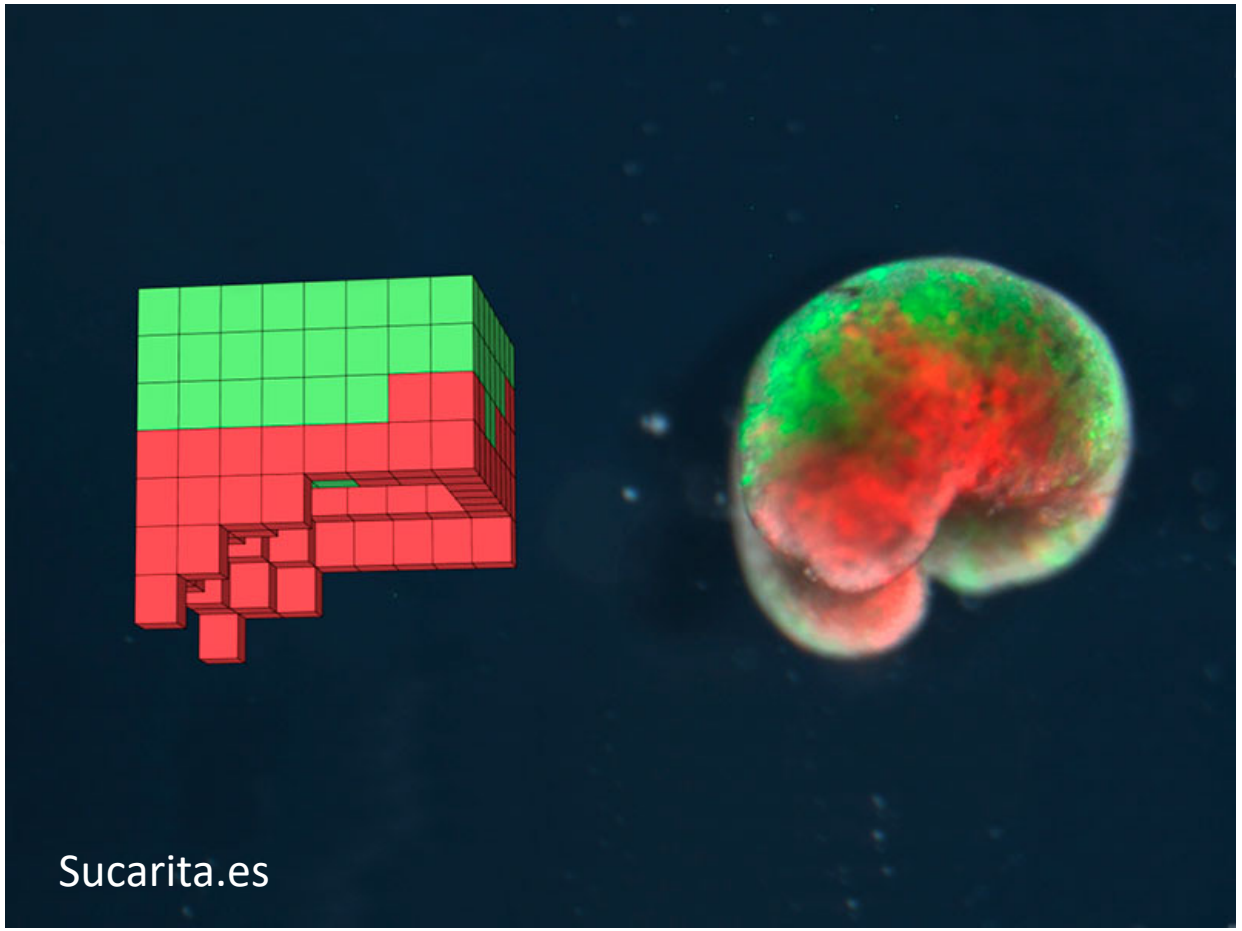


Mejora de rendimiento entre 20% y 30% en actividades masivas

Casos de aplicación

Dron acuático

Biobot



¡Gracias!

por ser parte de la comunidad Innovar +

Innovar + 2021
Desafíos que transforman vidas
23 al 27 de agosto

Grupo·epm