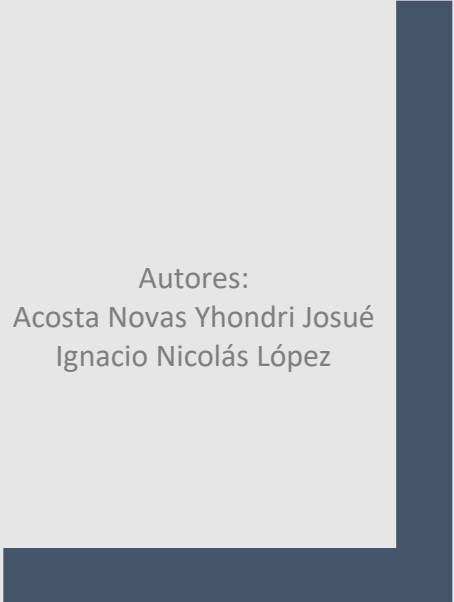




MEMORIA PRÁCTICA 2

Autores:
Acosta Novas Yhondri Josué
Ignacio Nicolás López



Implementación

Para la implementación de esta práctica hemos tomado el esqueleto de la práctica 1 como base. Como cambio destacable tenemos que la representación de los individuos ahora es una lista de enteros.

Este cambio lo hemos llevado a cabo para facilitarnos la tarea a la hora de hacer los cruces y mutaciones de la población.

Algoritmos de cruce implementados

- Cruce por ciclos (CX)
- Cruce por codificación ordinal
- Cruce por orden OX
- Cruce por recombinación de rutas
- PMX

Algoritmos de mutación implementados

- Mutación heurística
- Mutación por inserción
- Mutación por intercambio
- Mutación por inversión

Algoritmos de selección implementados (práctica 1)

- Ranking
- Ruleta
- Torneo
- Truncamiento
- Estocástico
- Restos

Problema Datos12

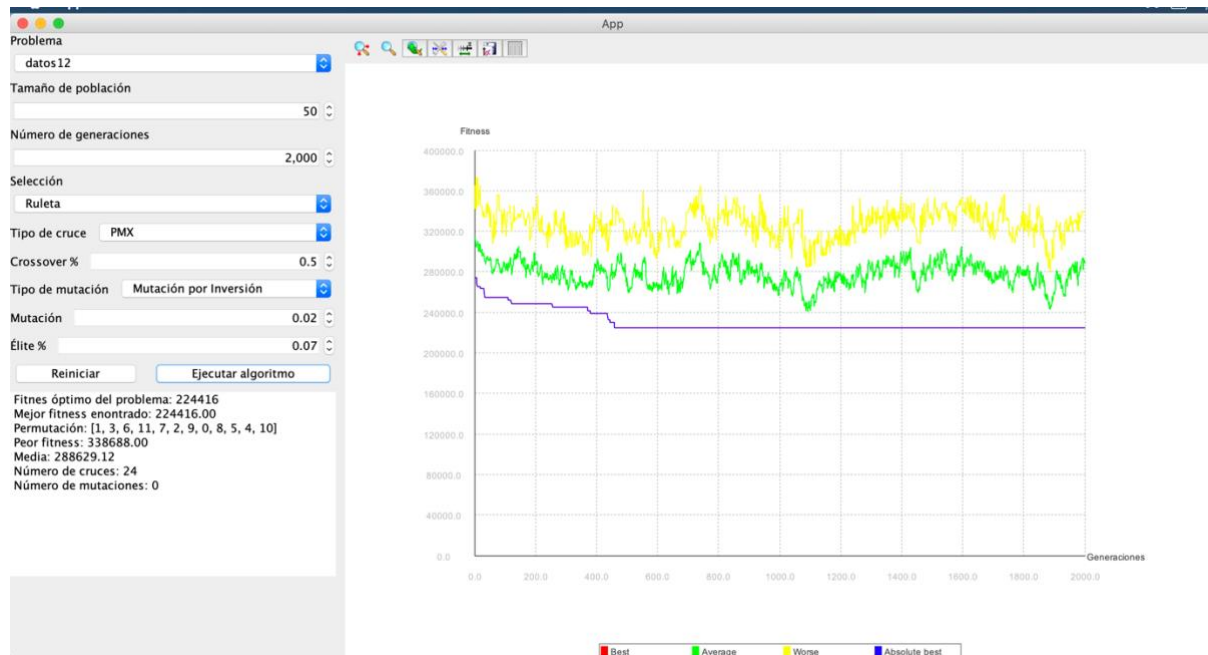


Imagen – Gráfico que ilustra el resultado óptima alcanzado y los parámetros necesarios

Tipo de cruce – PMX

Consiste en elegir un tramo de uno de los progenitores y cruzar preservando el orden y la posición de la mayor cantidad de ciudades del otro.

```
public Pair<PathChromosome, PathChromosome> crossover(PathChromosome chromosomeA, PathChromosome chromosomeB) {
    List<Integer> genesA = new ArrayList<>(Collections.nCopies(chromosomeA.getGenes().size(), 0));
    List<Integer> genesB = new ArrayList<>(Collections.nCopies(chromosomeA.getGenes().size(), 0));

    Set<Integer> positions = Utils.pickUniqueRandomList(2, chromosomeA.getGenes().size() - 1);
    Iterator<Integer> iterator = positions.iterator();
    int temp1, temp2;
    temp1 = iterator.next();
    temp2 = iterator.next();
    int startPoint = Math.min(temp1, temp2);
    int endPoint = Math.max(temp1, temp2);

    List<Integer> homologosA = new ArrayList<>();
    List<Integer> homologosB = new ArrayList<>();
    //1º Intercambiamos los genes del segmento seleccionado
    for (int i = startPoint; i < endPoint; i++) {
        genesA.set(i, chromosomeB.getGenes().get(i));
        genesB.set(i, chromosomeA.getGenes().get(i));
        homologosA.add(chromosomeA.getGenes().get(i));
        homologosB.add(chromosomeB.getGenes().get(i));
    }

    for (int i = 0; i < chromosomeA.getGenes().size(); i++) {
        //En este punto no intercambiamos los genes del segmento seleccionado anteriormente
        if (i >= startPoint && i < endPoint) {
            continue;
        }

        if (genesA.contains(chromosomeA.getGenes().get(i))) {
            for (int j = 0; j < homologosA.size(); j++) {
                if (genesA.contains(homologosA.get(j))) {
                    genesA.set(i, homologosA.get(j));
                    break;
                }
            }
        } else {
            genesA.set(i, chromosomeA.getGenes().get(i));
        }

        if (genesB.contains(chromosomeB.getGenes().get(i))) {
            for (int j = 0; j < homologosB.size(); j++) {
                if (genesB.contains(homologosB.get(j))) {
                    genesB.set(i, homologosB.get(j));
                    break;
                }
            }
        } else {
            genesB.set(i, chromosomeB.getGenes().get(i));
        }
    }

    return new Pair<>(new PathChromosome(genesA), new PathChromosome(genesB));
}
```

Imagen – Implementación del cruce PMX.

Mutación por inversión

Este tipo de mutación se aplica con una determinada probabilidad y consiste en seleccionar dos puntos el individuo al azar e invertir los elementos que hay entre dichos puntos.

```
@Override
public PathChromosome mutate(PathChromosome chromosome) {
    List<Integer> newGenes = chromosome.cloneGenotype();

    Pair<Integer, Integer> maxMinPair = Utils.getMaxMinPosition( maxElement: chromosome.getGenes().size()-1);
    int initialPoint = maxMinPair.getElement0();
    int endPoint = maxMinPair.getElement1();

    for (int i = initialPoint, j = endPoint; i < endPoint; i++, j--) {
        Collections.swap(newGenes, i, j);
    }

    return new PathChromosome(newGenes);
}
```

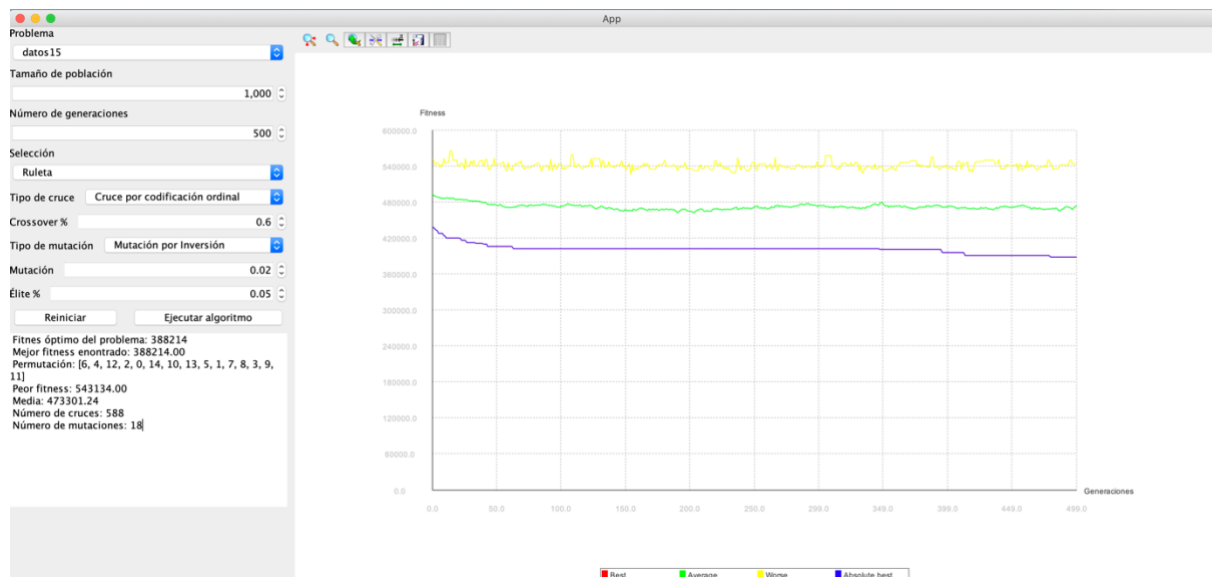
Imagen – Implementación mutación por inversión.

La siguiente tabla muestra los resultados de ejecutar 20 veces el problema Datos12 con los mismos parámetros.

1	Individuo	Mejor Fitness	Peor Fitness	Media
2	[1, 3, 6, 11, 7, 2, 9, 0, 8, 5, 4, 10]	224416.00	338688.00	288629.12
3	[4, 11, 8, 0, 1, 9, 5, 7, 10, 3, 2, 6]	235704.00	332254.00	263414.64
4	[4, 11, 9, 0, 1, 10, 3, 5, 8, 7, 2, 6]	233040.00	337752.00	291800.00
5	[5, 3, 9, 0, 7, 8, 6, 2, 1, 4, 11, 10]	237804.00	315952.00	279069.20
6	[7, 3, 8, 11, 1, 2, 9, 0, 6, 4, 5, 10]	229092.00	328866.00	295643.24
7	[4, 11, 9, 0, 1, 10, 3, 5, 8, 7, 2, 6]	233040.00	338954.00	293547.00
8	[10, 4, 3, 0, 1, 7, 8, 2, 9, 5, 11, 6]	240732.00	322206.00	279374.68
9	[1, 3, 6, 11, 7, 2, 9, 0, 8, 5, 4, 10]	224416.00	319218.00	267718.60
10	[10, 4, 3, 0, 1, 7, 8, 2, 9, 5, 11, 6]	240732.00	315926.00	268183.36
11	[2, 7, 9, 4, 3, 8, 6, 5, 10, 0, 11, 1]	246404.00	293148.00	263631.40
12	[5, 3, 9, 0, 7, 8, 6, 2, 1, 4, 11, 10]	237804.00	318680.00	280299.16
13	[5, 3, 9, 0, 7, 8, 6, 2, 1, 4, 11, 10]	237804.00	339496.00	291105.24
14	[10, 7, 3, 0, 8, 5, 9, 4, 6, 11, 2, 1]	234788.00	321190.00	265286.92
15	[1, 3, 6, 11, 7, 2, 9, 0, 8, 5, 4, 10]	224416.00	308368.00	260243.44
16	[11, 3, 6, 8, 5, 2, 4, 0, 7, 1, 9, 10]	242450.00	300154.00	267151.16
17	[4, 11, 9, 0, 1, 10, 3, 5, 8, 7, 2, 6]	233040.00	320602.00	273461.56
18	[10, 4, 3, 0, 6, 9, 5, 7, 8, 11, 2, 1]	232164.00	330686.00	286596.48
19	[9, 2, 10, 11, 5, 4, 1, 3, 8, 7, 0, 6]	237756.00	291254.00	256150.20
20	[3, 11, 10, 0, 1, 8, 4, 9, 5, 7, 2, 6]	234188.00	323500.00	275322.32
21	[1, 3, 6, 11, 7, 2, 9, 0, 8, 5, 4, 10]	224416.00	339052.00	286013.16

Destacar que podríamos haber disminuido el número de generaciones ya que como podemos apreciar en el gráfico de la solución, esta se consigue alrededor de la generación 550 – 600.

Problema Datos15



Cruce por codificación ordinal

Codificamos los genes de los individuos y después los cruzamos aplicando el cruce clásico. Por último descodificamos los genes.

```
public Pair<PathChromosome, PathChromosome> crossover(PathChromosome chromosomeA, PathChromosome chromosomeB) {
    List<Integer> encodedA = encode(chromosomeA.getGenes());
    List<Integer> encodedB = encode(chromosomeB.getGenes());

    Pair<List<Integer>, List<Integer>> crossoverResult = getSinglePointCrossover(encodedA, encodedB);
    List<Integer> genesA = decode(crossoverResult.getElement0());
    List<Integer> genesB = decode(crossoverResult.getElement1());

    return new Pair<>(new PathChromosome(genesA), new PathChromosome(genesB));
}

public Pair<List<Integer>, List<Integer>> getSinglePointCrossover(List<Integer> chromosomeA, List<Integer> chromosomeB) {
    int chromosomeSize = Math.max(chromosomeA.size(), chromosomeB.size());
    Random random = new Random();
    int crossoverPoint = random.nextInt( bound: chromosomeA.size()-1);

    List<Integer> chromosomeNew = new ArrayList<>(Collections.nCopies(chromosomeA.size(), @ null));
    List<Integer> chromosomeBNew = new ArrayList<>(Collections.nCopies(chromosomeA.size(), @ null));

    for (int i = 0; i < crossoverPoint; i++) {
        if (i < chromosomeNew.size() && i < chromosomeA.size()) {
            chromosomeNew.set(i, chromosomeA.get(i));
        }

        if (i < chromosomeBNew.size() && i < chromosomeB.size()) {
            chromosomeBNew.set(i, chromosomeB.get(i));
        }
    }

    for (int i = crossoverPoint; i < chromosomeSize; i++) {
        if (i < chromosomeNew.size() && i < chromosomeB.size()) {
            chromosomeNew.set(i, chromosomeB.get(i));
        }

        if (i < chromosomeBNew.size() && i < chromosomeA.size()) {
            chromosomeBNew.set(i, chromosomeA.get(i));
        }
    }

    return new Pair<>(chromosomeNew, chromosomeBNew);
}

private List<Integer> encode(List<Integer> parent) {
    List<Integer> newGenes = new ArrayList<>();
    List<Integer> dinamicList = new ArrayList<>();
    for (int i = 0; i < parent.size(); i++) {
        dinamicList.add(new Integer(i));
    }

    for (int i = 0; i < parent.size(); i++) {
        Integer value = parent.get(i);
        int newValue = dinamicList.indexOf(value);
        newGenes.add(newValue);
        dinamicList.remove(new Integer(value));
    }

    return newGenes;
}

private List<Integer> decode(List<Integer> parent) {
    List<Integer> newGenes = new ArrayList<>();
    List<Integer> dinamicList = new ArrayList<>();
    for (int i = 0; i < parent.size(); i++) {
        dinamicList.add(new Integer(i));
    }

    for (int i = 0; i < parent.size(); i++) {
        Integer value = parent.get(i);
        int newValue = dinamicList.get(value);
        newGenes.add(newValue);
        dinamicList.remove((int)value);
    }

    return newGenes;
}
```

Imágenes cruce por codificación ordinal.

La siguiente tabla muestra los resultados de ejecutar 20 veces el problema Datos15 con los mismos parámetros.

1	Individuo	Mejor Fitness	Peor Fitness	Media
2	[6, 4, 12, 2, 0, 14, 10, 13, 5, 1, 7, 8, 3, 9, 11]	388214.00	543134.00	473301.24
3	[8, 5, 7, 0, 13, 10, 3, 4, 1, 14, 2, 11, 6, 12, 9]	398118.00	565586.00	482596.88
4	[8, 1, 12, 0, 13, 3, 10, 6, 4, 5, 9, 2, 14, 11, 7]	402986.00	547334.00	480813.01
5	[6, 3, 11, 4, 7, 0, 1, 9, 2, 5, 13, 14, 8, 10, 12]	397716.00	552590.00	493731.44
6	[9, 3, 2, 4, 7, 0, 8, 1, 11, 10, 14, 13, 5, 6, 12]	391170.00	545392.00	475209.57
7	[10, 4, 7, 11, 12, 0, 8, 2, 5, 9, 3, 13, 6, 1, 14]	395388.00	547076.00	472199.34
8	[0, 7, 8, 9, 3, 2, 6, 14, 13, 5, 12, 1, 10, 11, 4]	392850.00	544442.00	475047.00
9	[1, 2, 13, 4, 11, 5, 3, 12, 6, 9, 14, 8, 10, 7, 0]	396116.00	546510.00	490141.01
10	[10, 9, 13, 11, 12, 2, 6, 7, 8, 0, 1, 5, 4, 14, 3]	393866.00	553710.00	469398.97
11	[3, 6, 7, 14, 0, 9, 8, 11, 4, 12, 5, 1, 10, 2, 13]	409000.00	547606.00	499123.69
12	[0, 10, 12, 8, 14, 1, 4, 7, 3, 11, 9, 5, 2, 13, 6]	405908.00	538400.00	474660.27
13	[8, 1, 7, 9, 12, 4, 6, 14, 13, 11, 3, 10, 0, 5, 2]	396718.00	534412.00	468932.13
14	[10, 2, 11, 0, 14, 4, 6, 1, 7, 5, 3, 8, 13, 9, 12]	391426.00	549062.00	476081.72
15	[14, 5, 2, 9, 13, 4, 6, 0, 3, 7, 11, 10, 1, 12, 8]	398368.00	545296.00	470386.01
16	[2, 10, 14, 9, 7, 0, 8, 1, 11, 3, 6, 13, 4, 12, 5]	403846.00	547900.00	473184.62
17	[9, 3, 12, 14, 11, 8, 6, 0, 10, 7, 5, 1, 4, 13, 2]	405390.00	545904.00	474013.03
18	[10, 4, 12, 14, 0, 2, 6, 7, 5, 1, 13, 8, 3, 9, 11]	391704.00	525526.00	466546.69
19	[10, 6, 12, 11, 14, 1, 4, 7, 8, 9, 0, 5, 2, 13, 3]	406704.00	541734.00	474022.57
20	[10, 2, 12, 0, 14, 4, 6, 1, 5, 7, 3, 13, 11, 9, 8]	392712.00	546536.00	467943.74
21	[7, 8, 6, 5, 12, 1, 4, 11, 9, 0, 2, 14, 10, 13, 3]	396516.00	549822.00	467754.49

En esta ocasión hemos tenido que aumentar considerablemente el tamaño de la población para poder conseguir el resultado buscado. Por último hemos ajustado correctamente el número de generaciones, ya que el resultado se consigue alrededor de la generación 450 y el número de generaciones total era de 500.

Problema Datos30

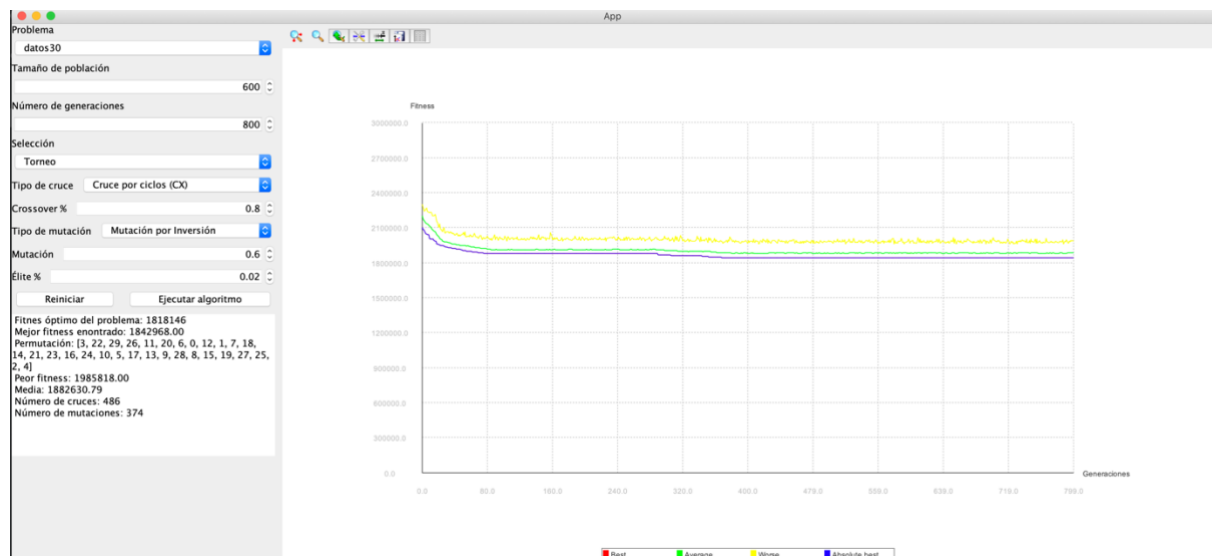


Imagen – Ilustra el ejercicio Datos30 donde hemos conseguido una aproximación del resultado.

Cruce por ciclos

1. Tomamos la primera posición nueva en el padre P1.
2. Buscamos el en la misma posición de P2.
3. Vamos a la posición con el mismo gen en P1.
4. Añadimos este gen al ciclo.
5. Repetimos los pasos 2-4 hasta que completemos el nuevo individuo.

```
@Override
public Pair<PathChromosome, PathChromosome> crossover(PathChromosome chromosomeA, PathChromosome chromosomeB) {
    List<Integer> genesA = getGenes(chromosomeA.getGenes(), chromosomeB.getGenes());
    List<Integer> genesB = getGenes(chromosomeB.getGenes(), chromosomeA.getGenes());
    return new Pair<>(new PathChromosome(genesA), new PathChromosome(genesB));
}

private List<Integer> getGenes(List<Integer> parentA, List<Integer> parentB) {
    List<Integer> newGenes = new ArrayList<>(Collections.nCopies(parentA.size(), 0));

    boolean end = false;
    int index = 0;
    int indexOfNextValue;
    Integer valueToInsert = parentA.get(index);
    newGenes.set(index, valueToInsert);
    indexOfNextValue = index;
    while (!end && indexOfNextValue < parentA.size()) {
        valueToInsert = parentB.get(indexOfNextValue);
        indexOfNextValue = parentA.indexOf(valueToInsert);
        if (!newGenes.contains(valueToInsert)) {
            newGenes.set(indexOfNextValue, valueToInsert);
        } else {
            end = true;
        }
    }

    for (int i = 0; i < newGenes.size(); i++) {
        if (newGenes.get(i) == null) {
            newGenes.set(i, parentB.get(i));
        }
    }

    return newGenes;
}
```

Imágenes cruce por ciclos.

La siguiente tabla muestra los resultados de ejecutar 20 veces el problema Datos30 con los mismos parámetros.

1	Individuo	Mejor Fitness	Peor Fitness	Media
2	[3, 22, 29, 26, 11, 20, 6, 0, 12, 1, 7, 18, 14, 21, 23, 16, 24, 10, 5, 17, 13, 9, 28, 8, 15, 19, 27, 25, 2, 4]	1842968.00	1985818.00	1882630.79
3	[5, 25, 22, 19, 14, 8, 0, 17, 24, 21, 18, 3, 29, 1, 20, 23, 9, 2, 6, 16, 27, 4, 26, 15, 7, 13, 10, 11, 12, 28]	1848088.00	1989180.00	1890850.40
4	[14, 22, 17, 11, 12, 23, 21, 7, 8, 24, 25, 27, 9, 19, 13, 5, 20, 1, 16, 18, 2, 6, 28, 3, 15, 29, 0, 26, 10, 4]	1854510.00	1979922.00	1891997.67
5	[10, 7, 1, 5, 26, 25, 23, 21, 9, 16, 11, 2, 29, 18, 20, 24, 22, 0, 13, 17, 28, 6, 15, 8, 19, 27, 4, 3, 12, 14]	1885032.00	2018178.00	1917733.19
6	[29, 6, 24, 2, 28, 22, 10, 26, 12, 18, 23, 3, 5, 0, 8, 14, 20, 11, 13, 17, 19, 15, 25, 7, 9, 1, 27, 4, 16, 21]	1882848.00	1991754.00	1917666.08
7	[1, 8, 16, 12, 21, 22, 17, 29, 9, 3, 2, 11, 5, 15, 10, 19, 14, 25, 7, 27, 0, 4, 23, 13, 26, 6, 18, 20, 24, 28]	1862664.00	2003036.00	1900571.44
8	[16, 5, 13, 20, 4, 10, 6, 17, 3, 9, 19, 21, 12, 24, 23, 26, 22, 7, 14, 18, 15, 11, 29, 0, 2, 27, 1, 8, 25, 28]	1912318.00	2033428.00	1944238.61
9	[19, 4, 12, 14, 28, 10, 13, 17, 1, 21, 7, 29, 15, 20, 9, 5, 2, 18, 6, 16, 23, 25, 8, 27, 26, 11, 3, 22, 24, 0]	1872846.00	1997942.00	1908980.11
10	[14, 2, 25, 20, 13, 12, 8, 24, 21, 17, 22, 26, 4, 28, 3, 0, 10, 29, 15, 16, 18, 7, 1, 19, 27, 9, 11, 23, 6, 5]	1903112.00	2003936.00	1935106.24
11	[22, 5, 16, 28, 8, 19, 13, 1, 3, 17, 25, 24, 18, 14, 23, 15, 10, 0, 4, 7, 27, 2, 29, 26, 11, 6, 20, 21, 12, 9]	1886396.00	2015920.00	1922013.30
12	[13, 20, 24, 28, 5, 19, 2, 7, 23, 1, 17, 8, 18, 3, 22, 16, 25, 0, 6, 9, 14, 21, 11, 29, 27, 15, 10, 26, 12, 4]	1893620.00	2024470.00	1926240.07
13	[8, 5, 1, 19, 15, 10, 23, 24, 11, 16, 20, 14, 7, 13, 17, 28, 3, 27, 9, 6, 29, 25, 2, 0, 4, 12, 18, 22, 26, 21]	1863442.00	1984530.00	1904325.81
14	[29, 10, 26, 8, 15, 18, 9, 12, 17, 0, 6, 28, 13, 11, 1, 27, 14, 25, 7, 21, 19, 2, 3, 20, 5, 16, 23, 24, 22, 4]	1845450.00	1987284.00	1885851.05
15	[17, 25, 23, 11, 28, 13, 0, 4, 16, 1, 18, 2, 15, 22, 3, 7, 6, 12, 14, 20, 24, 19, 21, 29, 5, 27, 9, 8, 10, 26]	1905026.00	2020824.00	1939199.85
16	[29, 28, 22, 4, 25, 19, 10, 16, 21, 1, 3, 0, 11, 15, 8, 18, 13, 20, 24, 17, 12, 26, 6, 23, 9, 27, 14, 2, 7, 5]	1868590.00	2007116.00	1902487.20
17	[24, 19, 25, 9, 7, 15, 11, 12, 22, 0, 28, 23, 2, 14, 17, 27, 6, 10, 20, 1, 21, 3, 5, 26, 4, 29, 16, 8, 13, 18]	1880850.00	1996868.00	1916655.94
18	[7, 6, 20, 9, 8, 27, 21, 10, 23, 0, 3, 16, 24, 15, 25, 22, 12, 17, 19, 29, 1, 14, 4, 13, 26, 28, 11, 5, 18, 2]	1904160.00	2021570.00	1937481.23
19	[1, 20, 12, 7, 25, 11, 17, 2, 26, 15, 22, 3, 5, 0, 8, 14, 27, 23, 9, 18, 29, 28, 4, 13, 24, 16, 10, 19, 6, 21]	1886324.00	2000830.00	1920323.52
20	[8, 11, 2, 4, 22, 23, 20, 9, 21, 17, 1, 13, 16, 5, 14, 19, 3, 27, 7, 12, 0, 25, 28, 24, 6, 10, 29, 18, 15, 26]	1865428.00	1988958.00	1901795.27
21	[9, 22, 23, 5, 11, 20, 21, 0, 10, 1, 2, 18, 13, 15, 25, 7, 19, 17, 4, 26, 14, 6, 12, 8, 29, 24, 27, 16, 3, 28]	1867586.00	2023472.00	1902091.09

Destacable: La curva de cambio a lo largo de las generaciones es bastante estable la diferencia entre el mejor y el peor valor a lo largo de las generaciones cada vez se hace más pequeña.

El mejor valor se consigue en una generación temprana, alrededor de la 350, por lo que aumentar el número de generaciones no ayudaría a conseguir una mejor aproximación del problema.

También probamos con conjunto de parámetros (algoritmo de selección, cruce, mutación diferentes), pero solo conseguíamos peores resultados de media que se alejaban muchísimo del resultado objetivo.

Otros algoritmos implementados:

Cruce YI:

Este algoritmo de cruce, crea un nuevo individuo cruzando los genes de forma aleatoria de ambos padres.

Mutación YI:

Esta mutación intercambia n posiciones elegidas al azar de los genes del padre.

Ajuntamos documento Excel que contiene las tablas con los datos de las ejecuciones de los 3 problemas.