UNIVERSIDAD DE GRANADA

MÁSTER DE CIENCIA DE DATOS E INGENIERÍA DE COMPUTADORES

CURSO ACADÉMICO 2019-2020

TÉCNICAS DE SOFT-COMPUTING PARA APRENDIZAJE Y OPTIMIZACIÓN.
REDES NEURONALES Y METAHEURÍSTICAS, PROGRAMACIÓN EVOLUTIVA Y BIOINSPIRADA.

Algoritmo Genético aplicado al problema de la Máxima Diversidad.

Implementación de Algoritmo Genético y ejecución de experimentos para la resolución del problema de la Máxima Diversidad.

Nicolás Cubero

7 de Junio de 2020

Índice

1. Introducción **3**
1.1. El Problema de la Diversidad Máxima 3
1.2. Algoritmos genéticos 3

2. Detalles de implementación **4**
2.1. Codificación y evaluación 4
2.2. Operadores implementados 5
2.2.1. Operador de Selección 5
2.2.2. Operador de cruce 6
2.2.3. Operador de mutación 7
2.3. Funcionamiento general del algoritmo 7

3. Experimentos realizados **9**
3.1. Instancia *GKD-c_1_n500_m50* 10
3.2. Instancia *GKD-c_2_n500_m50* 11
3.3. Instancia *GKD-c_3_n500_m50* 12
3.4. Instancia *GKD-c_4_n500_m50* 12
3.5. Instancia *GKD-c_5_n500_m50* 12
3.6. Instancia *GKD-c_6_n500_m50* 13
3.7. Instancia *GKD-c_7_n500_m50* 13
3.8. Instancia *GKD-c_8_n500_m50* 13
3.9. Instancia *GKD-c_9_n500_m50* 14
3.10. Instancia *GKD-c_10_n500_m50* 14
3.11. Instancia *MDG-a_21_n2000_m200* 14
3.12. Instancia *MDG-a_22_n2000_m200* 15
3.13. Instancia *MDG-a_23_n2000_m200* 15
3.14. Instancia *MDG-a_24_n2000_m200* 15
3.15. Instancia *MDG-a_25_n2000_m200* 16
3.16. Instancia *MDG-a_26_n2000_m200* 16
3.17. Instancia *MDG-a_27_n2000_m200* 16
3.18. Instancia *MDG-a_28_n2000_m200* 17
3.19. Instancia *MDG-a_29_n2000_m200* 17
3.20. Instancia *MDG-a_30_n2000_m200* 17
3.21. Instancia *SOM-b_11_n300_m90* 17
3.22. Instancia *SOM-b_12_n300_m120* 18
3.23. Instancia *SOM-b_13_n400_m40* 18
3.24. Instancia *SOM-b_14_n400_m80* 19
3.25. Instancia *SOM-b_15_n400_m120* 19
3.26. Instancia *SOM-b_16_n400_m160* 19

3.27. Instancia *SOM-b_17_n500_m50* 20
3.28. Instancia *SOM-b_18_n500_m100* 20
3.29. Instancia *SOM-b_19_n500_m150* 21
3.30. Instancia *SOM-b_20_n500_m150* 21

1. Introducción

En este proyecto se lleva a cabo la implementación del **Algoritmo Genético Evolutivo**, en lenguaje C++, para la resolución del **problema de la Máxima Diversidad**.

Asimismo, también se pretende elaborar un conjunto de experimentaciones sobre la ejecución de esta Metaheurística aplicado a diversas este problema y realizar un breve análisis de los resultados obtenidos de la ejecución de estos experimentos.

1.1. El Problema de la Diversidad Máxima

El problema de la diversidad máxima, trata de seleccionar de entre $N = \{e_1, ..., e_n\}$ objetos posibles, un subconjunto de $M = \{e_1, ..., e_m\}$ $(M \subset N)$ objetos tal que la **diversidad** entre los objetos de dicho subconjunto M sea máxima.

Para cada par de objetos i y j del conjunto de N objetos posibles, se define una **distancia** $d_{i,j} = d(e_i, e_j)$ como una medida de la "distinción" entre cadas par de objetos, de forma que la diversidad total del subconjunto M de objetos se define como:

$$\forall e_i, e_j \in M \,|\, j > i |\, z = \sum d(e_i, e_j) \tag{1}$$

Cada subconjunto M válido constituye una posible solución al problema.

1.2. Algoritmos genéticos

En líneas generales, los algoritmos genéticos trabajan de la siguiente forma:

Los algoritmos genéticos se tratan de un tipo de **algoritmo metaheurístico evolutivo generacional** que, inspirado los **postulados neodarwinistas** sobre la **Evolución de las Especies**, propone la generación de **sucesivas poblaciones de individuos** (soluciones a un problema dado) mediante la simulación de un proceso evolutivo, con la finalidad de generar una solución óptima al problema dado.

De este modo, partiendo de una **población inicial** generada, bien de forma aleatoria o siguiendo otra estrategia, simula el proceso evolutivo sobre las

soluciones de dicha población.

Dicho proceso evolutivo se basa en la aplicación ordenada de tres operadores:

En primer lugar, un operador de **selección** que selecciona un subconjunto de soluciones de la población de partida (*parents*), otorgando mayor probabilidad de selección a las mejores soluciones (soluciones mayor valor de la función objetivo).

A continuación, sobre el subconjunto de la población seleccionada, se aplica un operador de **cruce** entre pares de individuos seleccionados aleatoriamente, mediante el cual se recombinan fragmentos del par de soluciones entre sí para conformar una nueva población de individuos (*offspring*), descendientes de los anteriores.

Por último, sobre la población de descendientes se aplica un operador de **mutación** que, modifica de forma aleatoria las componentes de cada solución con una determinada probabilidad.

De este modo, la población resultante reemplaza a la población de partida y se ejecuta otra nueva generación con la aplicación ordenada de los anteriores operadores. El proceso se repite hasta que se satisface la condición de parada, y la mejor solución se toma de entre todas las poblaciones sucedidas.

2. Detalles de implementación

2.1. Codificación y evaluación

En esta implementación se codifica cada solución como un vector binario de longitud igual al conjunto total de objetos $X = \{x_1, ..., x_n\}$, de forma que cada objeto x_i posee valor 1 si se incluye en el subconjunto o 0 si no se introduce en el subconjunto.

A esta codificación se debe de añadir la siguiente **restricción**: El número de objetos con valor 1 en el vector X debe de ser igual al tamaño del subconjunto, en caso contrario, no constituye una solución válida.

$$\forall x_i \in X, x_i = \begin{cases} 1 & el\ objeto\ i\ se\ incluye \\ 0 & el\ objeto\ i\ no\ se\ incluye \end{cases}, s.a. \sum_{i=1}^{n} x_i = M \qquad (2)$$

De este modo, la puntuación (*fitness*) de cada solución dada la matriz de distancias $d_{i,j}$ se computa tal y como se describe a continuación. Asimismo, se configura el cálculo de del *fitness* para penalizar las soluciones no válidas, a las cuales se les asigna un valor nulo:

$$fitness = \begin{cases} \sum_{i=1}^{n} \sum_{j=i+1}^{n} x_i * xj * d_{i,j} & si \sum_{i=1}^{n} x_i = M \\ 0 & si \sum_{i=1}^{n} x_i \neq M \end{cases} \quad (3)$$

2.2. Operadores implementados

Para la resolución del problema propuesto con la codificación y restricciones vigentes, los operadores de selección, cruce y mutación del algoritmo genético se implementan como se describe en esta sección:

2.2.1. Operador de Selección

Como operador de selección, se implementa una **selección por torneo** (*Tournament Selector*), que genera una nueva población de soluciones de tamaño p mediante la ejecución de p torneos, en cada uno de los cuales se eligen aleatoriamente k soluciones de la población original y se selecciona la mejor de las k soluciones, que pasa a forma parte de los p individuos seleccionados.

La descripción algorítmica se expone a continuación:

Algoritmo 1 Construye una nueva población a partir de otra inicial seleccionando los individuos de la nueva solución mediante Tournament Selector

```
procedure TOURNAMENT_SELECTOR(population, p, k)
    parents ← ∅

    for all i ∈ {1, ..., p} do

        contestants ← ∅

        for all c ∈ {1, ..., k} do
            sol ← select_random_solution(population)
            contestants = contestants ∪ sol

        best ← select_best_solution(contestants)

        parents = parents ∪ best
    return parents
```

2.2.2. Operador de cruce

Se decide implementar un operador de cruce binario en un único punto que, dado un par de soluciones a cruzar, selecciona un punto común en ambas soluciones en el cual las soluciones se parten en dos trozos. A continuación, se une la cabeza de la primera solución con la cola de la segunda solución y la cabeza de la segunda solución con la cola de la primera solución.

La implementación realizada selecciona pares aleatorios de la población original, y con una probabilidad p_{cross} ejecuta el cruzamiento y coloca las soluciones generadas en la nueva población o copia las soluciones originales sin cruzar en la nueva población.

El proceso algorítmico se describe a continuación:

Algoritmo 2 Construye una nueva población a partir de otra inicial efectuando con una probabilidad cruzamiento en un punto de pares de soluciones seleccionadas aleatoriamente del conjunto original

procedure CROSSOVER_OPERATOR(population, p_{cross})
  $offspring \leftarrow \emptyset$

  $s \leftarrow \frac{length(population)}{2}$
  for all $i \in \{1, ..., s\}$ **do**

    $sol1 \leftarrow select\_random\_solution(population)$
    $sol2 \leftarrow select\_random\_solution(population)$

    if $rand() < p_{cross}$ **then**
      $x \leftarrow random\_index(sol1)$

      $h1, t1 \leftarrow split(sol1, x)$
      $h2, t2 \leftarrow split(sol2, x)$

      $sol1' \leftarrow h1 \cup t2$
      $sol2' \leftarrow h2 \cup t1$

      $offspring \leftarrow offspring \cup \{sol1', sol2'\}$
    else
      $offspring \leftarrow offspring \cup \{sol1, sol2\}$
  return offspring

Este operador presenta el inconveniente de que puede generar soluciones que no satisfacen la restricción de la codificación, para solventar este problema, se implementa un **mecanismo de correción** consistente en la inclusión o eliminación de objetos seleccionados de forma aleatoria en el subconjunto representado por la solución hasta que dicho subconjunto quede con el número de objetos marcado por la instancia del problema.

2.2.3. Operador de mutación

Se implementa un operador de mutación binario basado en el intercambio aleatorio de las componentes de la solución que, para componente de cada solución de la población, con una probabilidad P_{mut}, intercambia el valor de esta componente por el de otra componente de la misma solución elegida aleatoriamente.

Este proceso de intercambio aleatorio entre las componentes de una solución, permite añadir diversidad a la población de soluciones, al tiempo que garantiza el cumplimiento de la restricción para cada solución mutada.

El proceso algorítmico se detalla a continuación:

Algoritmo 3 Construye una nueva población a partir de otra inicial efectuando con una probabilidad cruzamiento en un punto de pares de soluciones seleccionadas aleatoriamente del conjunto original

```
procedure MUTATOR_OPERATOR(population, p_mut)
    for all s ∈ population do

        for all i ∈ {1, ..., n} do
            if rand() < p_mut then
                i' ← random_index(s)

                aux ← s_i
                s_i ← s_i'
                s_i' ← aux
```

2.3. Funcionamiento general del algoritmo

La ejecución del algoritmo genético comienza por la **generación de la población inicial** de forma **aleatoria**, de modo que cada solución de dicha población se construye introduciendo en el subconjunto representado por la

misma elementos seleccionados de forma aleatoria del conjunto total de objetos hasta completar la capacidad admitida para los subconjuntos.

Tras la generación de la población inicial se ejecuta de forma secuencial los procesos de selección de las mejores soluciones de la población inicial mediante el operador de selección por torneo, el cruzamiento genético entre pares de soluciones aleatorias del conjunto seleccionado según el operador de cruce definido y la mutación aleatoria de componentes de cada solución dando lugar a una nueva población descendiente (*offspring*) que reemplaza a la población general.

Este proceso se repite hasta que la condición de parada se cumple, acto seguido, se devuelve la mejor solución encontrada a lo largo de las sucesivas generaciones de poblaciones.

Esta condición de parada detendrá la ejecución del algoritmo en cualquiera de estas 3 situaciones:

- Se sobrepasa el tiempo máximo de ejecución permitido.
- Se ejecuta el número máximo de iteraciones (generaciones) del algoritmo permitido.
- Se sobrepasa el número máximo de evaluaciones (cálculo de *fitness*) de soluciones permitido.

El flujo general de ejecución del algoritmo quedaría de la siguiente forma:

Algoritmo 4 Aplicación de Genetic Algorithm sobre el problema de la Máxima Diversidad

procedure GENETIC_ALGORITHM(population_size, k, p_{cross}, p_{mut})
$population \leftarrow generate\_random\_population(population\_size)$

$fittestSolution \leftarrow select\_best\_solution(population)$
$bestSolution \leftarrow fittestSolution$

while *not* $stop\_condition\_is\_met()$ **do**

Calcular la puntuación *fitness* de cada solución
$\forall sol \in population, sol.evaluateFitness()$

Selección de mejores individuos de la población
$parents \leftarrow tournament\_selector(population, population\_size, k)$

Cruce genético aleatorio entre pares de individuos
$offspring \leftarrow crossover\_operator(parents, p_{cross})$

Mutación aletoria de los individuos de la descendencia
$mutator\_operator(offspring, p_{mut})$

La polación descendiente reemplaza a la original
$population \leftarrow offspring$

Tomar la mejor solución de la población descendiente
$fittestSolution \leftarrow select\_best\_solution(population)$
$bestSolution \leftarrow select\_best(bestSolution, fittestSolution)$

return *bestSolution*

3. Experimentos realizados

Se aplicó este algoritmo genético con la implementación desarrollada sobre un total de 30 instancias del problema de la Máxima Diversidad, realizando un total de 10 experimentos con diferentes semillas para cada una de las instancias obteniendo el rendimiento medio para cada una de las instancias:

La **condición de parada** establecida en los experimentos viene determinada por las siguientes condiciones:

- Tiempo máximo de ejecución: 60 segundos.
- Número máximo de generaciones: 700.000.
- Número máximo de evaluaciones de soluciones: 700.000

Por su parte, en todas los experimentos se mantuvieron constantes el siguiente conjunto de parámetros con los siguientes valores:

- Nº de candidatos considerados por el operador de selección por torneo (K): 6
- Tamaño de la población: 100.
- Número de individuos seleccionados por el operador de selección: 100.

  Al ser igual que el tamaño de la población, se asume que cada individuo de la población original puede ser seleccionado más de 1 vez.

El conjunto de semillas utilizadas son las siguientes: 12345678, 23456781, 34567812, 45678123, 56781234, 67812345, 78123456, 81234567, 12435678, 24356781.

Por último, se realizaron diferentes experimentaciones para cada instancia considerando diferentes valores de probabilidad de cruzamiento p_{cross} y de probabilidad de mutación p_{mut} para tratar de determinar su influencia en el rendimiento del algoritmo.

3.1. Instancia *GKD-c_1_n500_m50*

Este primer conjunto de instancias propone un conjunto total de 500 objetos y un tamaño de subconjunto de 50 objetos.

En la siguiente tabla se exponen el valor de *fitness* medio, el valor de *fitness* máximo, el valor de *fitness* mínimo y la desviación estándar respecto al valor medio de la mejor solución obtenida a lo largo de todos los experimentos con las diferentes semillas ante los diferentes valores de probabilidad p_{cross} y p_{mut}:

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17221.3	17113.3	17171.9	34.3622
	0.6	17207.9	17023.3	17124.6	49.3982
0.7	0.35	17450.1	17235.4	17325	67.3401
	0.6	17307.9	17057.8	17163.5	64.5197

Tabla 1: Fitness obtenido por GA sobre la instancia GKD-c_1_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

Se observa que los mejores resultados se obtienen cuando se establece una probabilidad de cruzamiento relativamente alta (0.6) y una probabilidad de mutación relativamente baja (0.35).

Los anteriores experimentos se refleja a grandes rasgos que, al incrementar la probabilidad de mutación p_{mut}, el rendimiento del algoritmo tiende a disminuir ligeramente mientras que al incrementar la probabilidad de cruce p_{cross}, el rendimiento del algoritmo tiende a aumentar.

3.2. Instancia *GKD-c_2_n500_m50*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17510.8	17235.4	17347	81.3396
	0.6	17308.3	17163.7	17230.5	57.1991
0.7	0.35	17689.6	17324.3	17424.6	96.3606
	0.6	17351.6	17247	17297.2	36.3617

Tabla 2: Fitness obtenido por GA sobre la instancia GKD-c_2_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

Del mismo modo que con la anterior instancia, los mejores rendimientos se obtienen con una probabilidad de cruzamiento de 0.6 y una probabilidad de mutación de 0.35.

Al igual que en la anterior experiementación, se observa que el incremento de la probabilidad de mutación lleva a un ligero empeoramiento en los resultados de los experimentos, mientras que un incremento de la probabilidad de cruzamiento, lleva a la obtención de mejores resultados en los experimentos realizados.

3.3. Instancia *GKD-c_3_n500_m50*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17293.9	17072.7	17197.7	78.8953
	0.6	17261.1	16972	17108.2	74.2874
0.7	0.35	17445.8	17193.5	17306.1	72.4974
	0.6	17229.1	17054.6	17117.5	56.0534

Tabla 3: Fitness obtenido por GA sobre la instancia GKD-c_3_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

3.4. Instancia *GKD-c_4_n500_m50*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17281.9	17138.4	17222.3	41.2359
	0.6	17467.3	16969.4	17134.6	135.615
0.7	0.35	17406.1	17207.9	17294.3	54.8222
	0.6	17224.8	17059.2	17151.5	46.4235

Tabla 4: Fitness obtenido por GA sobre la instancia GKD-c_4_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

3.5. Instancia *GKD-c_5_n500_m50*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17461.2	17182.3	17271.8	82.7304
	0.6	17374.8	17124.6	17188.4	70.7588
0.7	0.35	17472.8	17283.7	17363.9	51.6292
	0.6	17336.7	17167.6	17254.2	48.6868

Tabla 5: Fitness obtenido por GA sobre la instancia GKD-c_5_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

3.6. Instancia *GKD-c_6_n500_m50*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17254.9	17078.8	17180.8	60.7803
	0.6	17164.2	16922.7	: 17054.1	69.3473
0.7	0.35	17368.5	17183.9	17261.1	67.7152
	0.6	17133.7	17020.4	17084.8	35.6771

Tabla 6: Fitness obtenido por GA sobre la instancia GKD-c_6_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

3.7. Instancia *GKD-c_7_n500_m50*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17386.1	17185.2	17240.9	66.1551
	0.6	17360	17064.6	17179.6	101.296
0.7	0.35	17507.5	17264.4	17360.9	72.9028
	0.6	17293.8	17127.2	17205	42.7998

Tabla 7: Fitness obtenido por GA sobre la instancia GKD-c_7_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

3.8. Instancia *GKD-c_8_n500_m50*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17400.9	17169.5	17321.3	69.0637
	0.6	17340.7	17063.6	17185.8	75.7829
0.7	0.35	17418.2	17262.9	17340.7	48.6609
	0.6	17475.3	17126.8	17271.2	92.7756

Tabla 8: Fitness obtenido por GA sobre la instancia GKD-c_8_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

3.9. Instancia *GKD-c_9_n500_m50*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17168.9	16892.3	17031.5	73.4332
	0.6	17138.3	16846.3	16943.9	98.2945
0.7	0.35	17292.8	17007.7	17131.9	87.0753
	0.6	17046.5	16853.6	16948.8	54.0081

Tabla 9: Fitness obtenido por GA sobre la instancia GKD-c_9_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

3.10. Instancia *GKD-c_10_n500_m50*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	17334	17141.7	17246.3	56.8861
	0.6	17266.1	17021.6	17121.5	74.9429
0.7	0.35	17410.1	17262.7	17343.7	46.8562
	0.6	17271.6	17086	17173.1	53.8984

Tabla 10: Fitness obtenido por GA sobre la instancia GKD-c_10_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

3.11. Instancia *MDG-a_21_n2000_m200*

Este nuevo conjunto de instancias propone un conjunto total de 2.000 objetos y un tamaño de subconjunto de 200 objetos.

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	101495	101211	101352	93.5759
	0.6	101321	101108	101199	66.3373
0.7	0.35	101641	101240	101424	104.184
	0.6	101663	101215	101431	126.921

Tabla 11: Fitness obtenido por GA sobre la instancia MDG-a_21_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

Los resultados obtenidos de la ejecución de los experimentos sobre esta instancia, revelan comportamientos similares a los obtenidos con las anteriores instancias: Al incrementar la probabilidad de cruce (p_{cross}) el rendimiento del algoritmo tiende a aumentar, mientras que al incrementar la probabilidad

de mutación (p_{mut}), el rendimiento disminuye.

De este modo, las mejores ejecuciones se obtienen con una probabilidad de cruce de 0.6 y una probabilidad de mutación de 0.35.

3.12. Instancia *MDG-a_22_n2000_m200*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	101408	101171	101265	72.706
	0.6	101512	101229	101400	85.3302
0.7	0.35	101512	101229	101400	85.3302
	0.6	101609	101118	101299	134.56

Tabla 12: Fitness obtenido por GA sobre la instancia MDG-a_22_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

3.13. Instancia *MDG-a_23_n2000_m200*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	101701	101164	101396	150.485
	0.6	101426	100948	101157	130.282
0.7	0.35	101653	101278	101461	116.296
	0.6	101559	101189	101324	117.522

Tabla 13: Fitness obtenido por GA sobre la instancia MDG-a_23_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

3.14. Instancia *MDG-a_24_n2000_m200*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	101465	101222	101349	82.9074
	0.6	101471	101170	101309	95.8999
0.7	0.35	101672	101279	101472	121.383
	0.6	101514	: 101077	101319	113.015

Tabla 14: Fitness obtenido por GA sobre la instancia MDG-a_24_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

3.15. Instancia *MDG-a_25_n2000_m200*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	101537	101175	101358	97.0761
	0.6	101816	101166	101320	178.254
0.7	0.35	101621	101395	101495	61.6945
	0.6	101611	101261	101414	98.6187

Tabla 15: Fitness obtenido por GA sobre la instancia MDG-a_25_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

3.16. Instancia *MDG-a_26_n2000_m200*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	101377	101154	101297	66.4651
	0.6	101582	101152	101300	136.978
0.7	0.35	101723	101209	101462	153.046
	0.6	101610	101130	101300	145.007

Tabla 16: Fitness obtenido por GA sobre la instancia MDG-a_26_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

3.17. Instancia *MDG-a_27_n2000_m200*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	101642	101140	101348	162.231
	0.6	101436	101101	101230	110.585
0.7	0.35	101777	101209	101391	145.41
	0.6	101471	101115	101299	100.969

Tabla 17: Fitness obtenido por GA sobre la instancia MDG-a_27_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

3.18. Instancia *MDG-a_28_n2000_m200*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	101568	101276	101390	100.213	
	0.6	101401	101107	101258	84.3
0.7	101797	101264	101471	165.528	
	0.6	101605	101209	101320	116.335

Tabla 18: Fitness obtenido por GA sobre la instancia MDG-a_28_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

3.19. Instancia *MDG-a_29_n2000_m200*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	101518	101147	101302	104.634
	0.6	101606	101099	101314	164.063
0.7	0.35	101762	101305	101446	122.002
	0.6	101650	101159	101375	136.033

Tabla 19: Fitness obtenido por GA sobre la instancia MDG-a_29_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

3.20. Instancia *MDG-a_30_n2000_m200*

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	101501	101137	101283	104.702
	0.6	101482	101065	101262	115.962
0.7	0.35	101596	101283	101394	94.2261
	0.6	101565	101102	101322	118.422

Tabla 20: Fitness obtenido por GA sobre la instancia MDG-a_30_n2000_m200 ante diferentes probabilidades de cruzamiento y mutación

3.21. Instancia *SOM-b_11_n300_m90*

A continuación, se ejecutan experimentos sobre un nuevo conjunto de instancias que proponen un número de total de objetos de 300 y un tamaño de subconjunto de 90 objetos:

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	18885	18823	18846	21.114
	0.6	18860	18745	18817.4	29.964
0.7	0.35	18923	18832	18880.6	28.0614
	0.6	18883	18750	18821.9	40.6607

Tabla 21: Fitness obtenido por GA sobre la instancia SOM-b_11_n300_m90 ante diferentes probabilidades de cruzamiento y mutación

Nuevamente, se aprecia que al incrementar la probabilidad de cruce, el rendimiento de los experiemntos tiende a aumentar de manera notable, mientras que al incrementar la probabilidad de mutación, los rendimientos muestran un empeoramiento.

De este modo, y al igual que en los experimentos realizados con el anterior conjunto de instancias, los mejores resultados se han obtenido con una probabilidad de cruce de 0.6 y una probabilidad de mutación de 0.35.

3.22. Instancia *SOM-b_12_n300_m120*

En esta instancia, se consideraron al igual que en la anterior, un conjunto total de 300 instancias pero un tamaño de subconjunto de 120.

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	33351	33234	33279.9	40.6287
	0.6	33306	33161	33240.9	46.1464
0.7	0.35	33414	33263	33347.8	53.1089
	0.6	33360	33189	33263.1	51.7135

Tabla 22: Fitness obtenido por GA sobre la instancia SOM-b_12_n300_m120 ante diferentes probabilidades de cruzamiento y mutación

3.23. Instancia *SOM-b_13_n400_m40*

Para esta instancia, se consideran un conjunto total de 400 instancias y un tamaño de subconjunto de 40.

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	3908	3855	3879.9	15.2082
	0.6	3968	3859	3883.8	30.763
0.7	0.35	3940	3884	3908.7	16.7036
	0.6	3919	3859	3890.4	21.1244

Tabla 23: Fitness obtenido por GA sobre la instancia SOM-b_13_n400_m40 ante diferentes probabilidades de cruzamiento y mutación

3.24. Instancia *SOM-b_14_n400_m80*

En esta instancia, se consideran al igual que en la anterior, un conjunto total de 400 instancias pero un tamaño de subconjunto de 80.

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	15043	14953	14989.3	24.0044
	0.6	15031	14941	14983.8	24.5512
0.7	0.35	15054	15006	15023	16.9588
	0.6	15033	14905	14981.8	36.4137

Tabla 24: Fitness obtenido por GA sobre la instancia SOM-b_14_n400_m80 ante diferentes probabilidades de cruzamiento y mutación

3.25. Instancia *SOM-b_15_n400_m120*

En esta instancia, se consideran al igual que en las anteriores, un conjunto total de 400 instancias pero se aumenta el tamaño de subconjunto a 120 objetos.

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	33352	33138	33265.9	61.2755
	0.6	33321	33160	33229.4	54.4614
0.7	0.35	33399	33221	33307.1	65.5385
	0.6	33304	33177	33225.5	35.0607

Tabla 25: Fitness obtenido por GA sobre la instancia SOM-b_15_n400_m120 ante diferentes probabilidades de cruzamiento y mutación

3.26. Instancia *SOM-b_16_n400_m160*

En esta instancia, se consideran al igual que en las anteriores, un conjunto total de 400 instancias pero se aumenta el tamaño de subconjunto a 160

objetos.

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	58670	58518	58585	52.8337
	0.6	58623	58434	58554.7	69.0812
0.7	0.35	58740	58604	58684.4	48.7426
	0.6	58689	58478	58600.6	54.7909

Tabla 26: Fitness obtenido por GA sobre la instancia SOM-b_16_n400_m160 ante diferentes probabilidades de cruzamiento y mutación

3.27. Instancia *SOM-b_17_n500_m50*

Esta instancia considera un conjunto total de 500 objetos y un subconjunto de tamaño 50:

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	6071	5945	5977.4	34.4302
	0.6	6036	5930	5977.8	32.514
0.7	0.35	6030	5967	5997	19.0683
	0.6	6064	5954	5992.9	33.848

Tabla 27: Fitness obtenido por GA sobre la instancia SOM-b_17_n500_m50 ante diferentes probabilidades de cruzamiento y mutación

3.28. Instancia *SOM-b_18_n500_m100*

Esta instancia considera, al igual que en la anterior, un conjunto total de 500 objetos, y un subconjunto de tamaño 100:

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	23295	23179	23236	36.4088
	0.6	23294	23137	23212	48.4665
0.7	0.35	23358	: 23208	23280.6	41.6802
	0.6	23301	23181	23230.4	38.0873

Tabla 28: Fitness obtenido por GA sobre la instancia SOM-b_18_n500_m100 ante diferentes probabilidades de cruzamiento y mutación

3.29. Instancia *SOM-b_19_n500_m150*

En esta instancia, el tamaño de subconjunto es incrementado a 150 objetos:

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	51826	51609	51746	68.3125
	0.6	51775	51522	51647.4	81.1359
0.7	0.35	51938	51716	51783.1	64.6799
	0.6	51834	51598	51719	76.1551

Tabla 29: Fitness obtenido por GA sobre la instancia SOM-b_19_n500_m150 ante diferentes probabilidades de cruzamiento y mutación

3.30. Instancia *SOM-b_20_n500_m150*

Por último, en esta instancia, el tamaño de subconjunto ha sido aumentado a 150 objetos:

p_{cross}	p_{mut}	Fitness máximo	Fitness mínimo	Fitness medio	Desviación típica
0.35	0.35	91297	91069	91168.1	71.1877
	0.6	91115	91027	91065.8	31.619
0.7	0.35	91415	91200	91305.7	75.1439
	0.6	91261	91033	91159.2	67.6148

Tabla 30: Fitness obtenido por GA sobre la instancia SOM-b_20_n500_m150 ante diferentes probabilidades de cruzamiento y mutación

A lo largo de los experimentos realizados sobre todas las instancias, se puede apreciar el mismo comportamiento en todas las experimentaciones, y que ya ha sido comentado anteriormente:

El incremento en la probabilidad de cruce y con ello el aumento del número de recombinaciones genéticas realizadas entre las mejores soluciones seleccionadas, permite incrementar notablemente los resultados obtenidos. De este modo, el operador de cruce implementado, facilitaría la construcción de óptimas y mejores soluciones a partir de las anteriores al estar constituidas por una recombinación de las ya existentes.

Por el contrario, los experimentos muestran un decaeimiento en los rendimientos de los algoritmos al incrementar la probabilidad de mutación. De este modo, la estrategia introducida por el operador de mutación, llevaría

en la mayoría de los casos a un empeoramiento del *fitness* de las soluciones mutadas.

En cualquier caso, si se comparan los rendimientos obtenidos de la aplicación de esta implementación del algoritmo genético con los resultados óptimos globales conocidos para estas instancias, se aprecia en todos los casos, que **los rendimientos obtenidos en los experimentos se aproximan en gran medida a los resultados óptimos globales**, lo cual nos lleva a justificar y a concluir que el algoritmo genético está funcionando y permite obtener soluciones óptimas al problema dado.