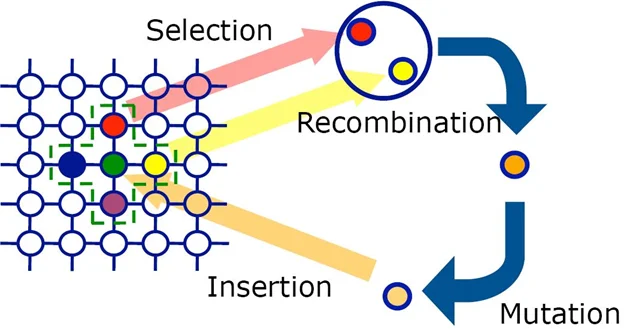
Algoritmos Genéticos



Adrián Yared Armas de la Nuez

**Contenido**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[**1. Enunciado 2**](#_2c35568x6wg8)

[**2. Explicación del código por partes 3**](#_mptzybqzhbp3)

[**2.1 Datos de Películas y Restricciones 3**](#_rjpj17ru25h3)

[**2.1.1 Código 3**](#_7xu8cnmv0vrh)

[**2.1.1 Explicación 3**](#_tk7vl0gmd99n)

[**2.2 Tamaños de DVD 4**](#_jdteuhpbbut0)

[**2.2.1 Código 4**](#_3ox9pexjuk2w)

[**2.2.1 Explicación 4**](#_9tx8mgywjupc)

[**2.3 Parámetros del Algoritmo Genético 4**](#_mnbxgyw3i3bi)

[**2.3.1 Código 4**](#_8y2s01w1f1bt)

[**2.3.1 Explicación 4**](#_evvcegk1yi3p)

[**2.4 Función calculate\_fitness 4**](#_kyc4kuagxw2g)

[**2.4.1 Código 4**](#_mr0wfuxww9cn)

[**2.4.1 Explicación 5**](#_7q7mslbkiuor)

[**2.5 Función create\_individual 5**](#_ggnyd94q5sa9)

[**2.5.1 Código 5**](#_cjm4wbijc9rg)

[**2.5.1 Explicación 5**](#_1du9oz7z3676)

[**2.6 Función Selection 6**](#_i5gfb2842m9c)

[**2.6.1 Código 6**](#_1rjy0h2qatqi)

[**2.6.1 Explicación 6**](#_59c8sfscm2fe)

[**2.7 Función Crossover 6**](#_g2q7sj3li11k)

[**2.7.1 Código 6**](#_1zt0dd6cgvp0)

[**2.7.1 Explicación 6**](#_iy4oa6ap2x7r)

[**2.8 Función Mutation 7**](#_2bu2hrmkyp9y)

[**2.8.1 Código 7**](#_gsdsi6p2zdw1)

[**2.8.1 Explicación 7**](#_jur9534x9ox3)

[**2.9 Función genetic\_algorithm 7**](#_pt9go7qh5im4)

[**2.9.1 Código 7**](#_su2ikuzacslw)

[**2.9.1 Explicación 8**](#_ni5tr1jxquf8)

[**2.10 Ejecución por cada tamaño de DVD 8**](#_xzx73rco28nr)

[**2.10.1 Código 8**](#_15f4yrizd834)

[**2.10.1 Explicación 8**](#_4c4tsqa796ia)

[**3. Resultado 8**](#_ph708nh4w8i)

[**4. Enlace a Colab 9**](#_98dxdlr3y4b8)

[**5. Código completo 9**](#_vqj4bhgw0gb2)

[**6. Bibliografía 12**](#_wqoaz875djju)

## **1. Enunciado**

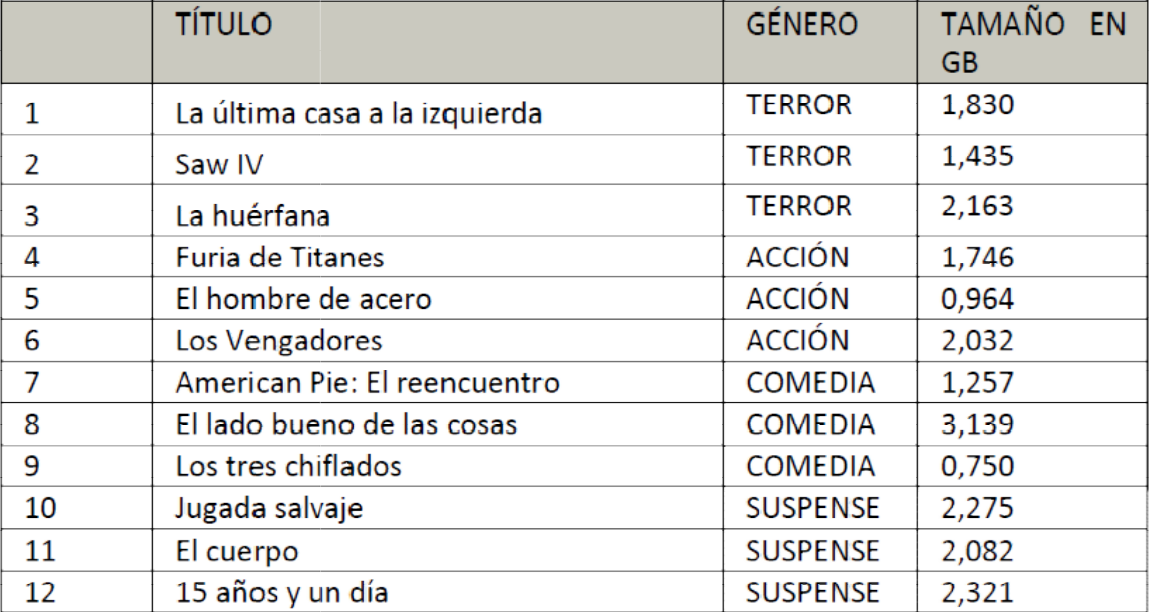
La tarea consiste en encontrar el subconjunto de películas cuyo tamaño mejor se ajusta a uno pedido (sin pasarse).

Se implementará una aplicación en Python de un algoritmo GENÉTICO que BUSQUE SOLUCIONES para grabar películas en DVD´s de DISTINTOS TAMAÑOS realizando el mejor ajuste para no desaprovechar el espacio de almacenamiento.

(Se probará con los tamaños 4.7, 8.5, 13.3 Y 15.9 GBytes mostrando las mejores soluciones obtenidas).

Las películas están catalogadas por distintos géneros, ACCIÓN, TERROR, COMEDIA y SUSPENSE.

* En un mismo DVD no se podrán mezclar películas de COMEDIA y TERROR.
* Tampoco se quiere poner en el mismo DVD las películas Jugada salvaje y El cuerpo ni Furia de Titanes y El hombre de acero.



Criterios de calificación:

(4 Puntos) El algoritmo muestra soluciones aceptables

(4 Puntos) Se crean y comentan los métodos de mutación, fitness, y recombinación.

(2 Puntos) Uso del vocabulario correcto (población, generación, cromosoma etc) así como explicación de los hiperparámetos (probabilidad de recombinación, de mutación, población inicial etc).

## **2. Explicación del código por partes**

### **2.1 Datos de Películas y Restricciones**

#### **2.1.1 Código**

# List of movies with attributes: title, genre, and size in GB.

movies = [

{"title": "The Last House on the Left", "genre": "HORROR", "size": 1.830},

{"title": "Saw IV", "genre": "HORROR", "size": 1.435},

{"title": "Orphan", "genre": "HORROR", "size": 2.163},

{"title": "Clash of the Titans", "genre": "ACTION", "size": 1.746},

{"title": "Man of Steel", "genre": "ACTION", "size": 0.964},

{"title": "The Avengers", "genre": "ACTION", "size": 2.032},

{"title": "American Reunion", "genre": "COMEDY", "size": 1.257},

{"title": "Silver Linings Playbook", "genre": "COMEDY", "size": 3.139},

{"title": "The Three Stooges", "genre": "COMEDY", "size": 0.750},

{"title": "Wild Card", "genre": "THRILLER", "size": 2.275},

{"title": "The Body", "genre": "THRILLER", "size": 2.082},

{"title": "15 Years and One Day", "genre": "THRILLER", "size": 2.321}

]

# Restrictions: certain movies cannot be selected together, represented as pairs.

movie\_restrictions = [

("Wild Card", "The Body"),

("Clash of the Titans", "Man of Steel")

]

# Genre restrictions: restrict selecting both genres in a single DVD.

genre\_restrictions = ("COMEDY", "HORROR")

#### **2.1.1 Explicación**

Esta es la lista de películas dictadas por el enunciado e introducidas en una lista de diccionarios.

**movie\_restrictions**: Algunas parejas de películas no pueden estar en el mismo DVD.

**genre\_restrictions**: Evita que Comedia y Terror estén juntos en el mismo DVD

### **2.2 Tamaños de DVD**

#### **2.2.1 Código**

# Available DVD sizes in GB.

dvd\_sizes = [4.7, 8.5, 13.3, 15.9]

#### **2.2.1 Explicación**

Esta lista contiene las capacidades en GB de diferentes tamaños de DVD.

### **2.3 Parámetros del Algoritmo Genético**

#### **2.3.1 Código**

# Genetic algorithm parameters.

NUM\_INDIVIDUALS = 100 # Population size

GENERATIONS = 100 # Number of generations

MUTATION\_PROBABILITY = 0.1 # Probability of mutation

#### **2.3.1 Explicación**

**NUM\_INDIVIDUOS**: Número de individuos en cada generación.

**GENERACIONES**: Cantidad de generaciones a evaluar.

**PROB\_MUTACION**: Probabilidad de que ocurra una mutación en un individuo

### **2.4 Función calculate\_fitness**

#### **2.4.1 Código**

def calculate\_fitness(individual, dvd\_size):

# Total size of selected movies.

total\_size = sum(m["size"] for i, m in enumerate(movies) if individual[i] == 1)

# If total size exceeds DVD capacity, return a fitness score of 0.

if total\_size > dvd\_size:

return 0

# Check genre restrictions: no selection should include both restricted genres.

genres = {movies[i]["genre"] for i in range(len(movies)) if individual[i] == 1}

if genre\_restrictions[0] in genres and genre\_restrictions[1] in genres:

return 0

# Check pair restrictions: restricted pairs should not be selected together.

for restriction in movie\_restrictions:

if any(m["title"] == restriction[0] for i, m in enumerate(movies) if individual[i] == 1) and \

any(m["title"] == restriction[1] for i, m in enumerate(movies) if individual[i] == 1):

return 0

return total\_size # Return the total size if all restrictions are met.

#### 

#### **2.4.1 Explicación**

Esta función evalúa a los individuos y si se pasa del máximo su valor fitness será 0.

En total\_size calcula el tamaño total de las películas con un 1, si se supera el tamaño se pone 0, además comprueba que no se mezclen géneros o títulos prohibidos por el enunciado.

### **2.5 Función create\_individual**

#### **2.5.1 Código**

def create\_individual():

# Create a random individual (a random binary selection of movies).

return [random.randint(0, 1) for \_ in movies]

#### **2.5.1 Explicación**

Crea un individuo aleatorio con una lista binaria en la que cada posición representa si una película está seleccionada (1) o no (0).

### **2.6 Función Selection**

#### **2.6.1 Código**

def selection(population, dvd\_size):

# Calculate fitness for each individual and sort by fitness.

scores = [(ind, calculate\_fitness(ind, dvd\_size)) for ind in population]

scores = sorted(scores, key=lambda x: x[1], reverse=True)

# Select top 50% individuals for mating.

selected = [s[0] for s in scores[:NUM\_INDIVIDUALS // 2]]

return selected

#### 

#### **2.6.1 Explicación**

Selecciona los mejores individuos de la población basándose en su fitness, para ello calcula el fitness de cada individuo, a continuación ordena la población de mayor a menor fitness y finalmente, selecciona la mitad superior de la población para la próxima generación.

### **2.7 Función Crossover**

#### **2.7.1 Código**

def crossover(ind1, ind2):

# Single-point crossover between two parents.

point = random.randint(1, len(ind1) - 2)

return ind1[:point] + ind2[point:], ind2[:point] + ind1[point:]

#### **2.7.1 Explicación**

Realiza el cruce entre dos individuos (padres) en un punto aleatorio, generando dos nuevos individuos (hijos). Para ello escoge un punto de cruce al azar e intercambia los genes (películas seleccionadas) de ambos padres desde ese punto para crear dos hijos.

### **2.8 Función Mutation**

#### **2.8.1 Código**

def mutation(individual):

# Apply mutation with a given probability.

if random.random() < MUTATION\_PROBABILITY:

point = random.randint(0, len(individual) - 1)

individual[point] = 1 - individual[point] # Flip the binary bit at the mutation point.

#### **2.8.1 Explicación**

Aplica una mutación aleatoria a un individuo para agregar variabilidad.

Si ocurre la mutación (de acuerdo a MUTATION\_PROBABILITY, cambia el estado de un gen aleatorio (0 a 1 o viceversa).

### **2.9 Función genetic\_algorithm**

#### **2.9.1 Código**

def genetic\_algorithm(dvd\_size):

# Initialize population with random individuals.

population = [create\_individual() for \_ in range(NUM\_INDIVIDUALS)]

for \_ in range(GENERATIONS):

# Selection of the top individuals for reproduction.

selected = selection(population, dvd\_size)

new\_population = []

# Generate new individuals by crossover and mutation.

while len(new\_population) < NUM\_INDIVIDUALS:

parent1 = random.choice(selected)

parent2 = random.choice(selected)

child1, child2 = crossover(parent1, parent2)

mutation(child1)

mutation(child2)

new\_population.extend([child1, child2])

population = new\_population # Update population with the new generation.

# Find the best individual with the highest fitness value.

best\_individual = max(population, key=lambda ind: calculate\_fitness(ind, dvd\_size))

best\_fitness = calculate\_fitness(best\_individual, dvd\_size)

# Print the best solution for the given DVD size.

selected\_titles = [movies[i]["title"] for i in range(len(movies)) if best\_individual[i] == 1]

print(f"Best fit for DVD of {dvd\_size} GB:")

print(f"Selected movies: {selected\_titles}")

print(f"Used space: {best\_fitness:.3f} GB\n")

#### **2.9.1 Explicación**

Ejecuta el ciclo completo del algoritmo genético, para ello inicializa una población aleatoria de individuos, a continuación para cada generación, selecciona los mejores individuos, luego realiza cruces y mutaciones para generar una nueva población y al final de todas las generaciones, identifica el mejor individuo de la población (el que usa mejor el espacio en el DVD sin violar restricciones) y para terminar imprime las películas seleccionadas para el tamaño de DVD dado.

### **2.10 Ejecución por cada tamaño de DVD**

#### **2.10.1 Código**

# Run the genetic algorithm for each DVD size.

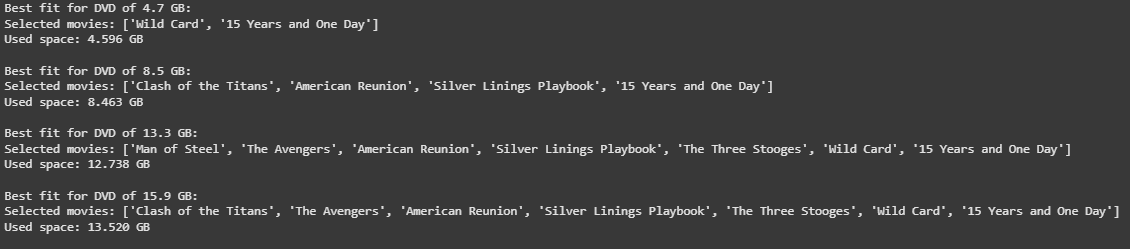
for size in dvd\_sizes:

genetic\_algorithm(size)

#### **2.10.1 Explicación**

Ejecuta el algoritmo genético para cada tamaño de DVD de la lista e imprime el mejor ajuste para cada uno.

## **3. Resultado**



## **4. Enlace a Colab**

[](https://colab.research.google.com/drive/1ICqDpBvX5nudeDTWNp7jfKZ4uiZPq1JC?usp=sharing)

## **5. Código completo**

import random

import numpy as np

# List of movies with attributes: title, genre, and size in GB.

movies = [

{"title": "The Last House on the Left", "genre": "HORROR", "size": 1.830},

{"title": "Saw IV", "genre": "HORROR", "size": 1.435},

{"title": "Orphan", "genre": "HORROR", "size": 2.163},

{"title": "Clash of the Titans", "genre": "ACTION", "size": 1.746},

{"title": "Man of Steel", "genre": "ACTION", "size": 0.964},

{"title": "The Avengers", "genre": "ACTION", "size": 2.032},

{"title": "American Reunion", "genre": "COMEDY", "size": 1.257},

{"title": "Silver Linings Playbook", "genre": "COMEDY", "size": 3.139},

{"title": "The Three Stooges", "genre": "COMEDY", "size": 0.750},

{"title": "Wild Card", "genre": "THRILLER", "size": 2.275},

{"title": "The Body", "genre": "THRILLER", "size": 2.082},

{"title": "15 Years and One Day", "genre": "THRILLER", "size": 2.321}

]

# Restrictions: certain movies cannot be selected together, represented as pairs.

movie\_restrictions = [

("Wild Card", "The Body"),

("Clash of the Titans", "Man of Steel")

]

# Genre restrictions: restrict selecting both genres in a single DVD.

genre\_restrictions = ("COMEDY", "HORROR")

# Available DVD sizes in GB.

dvd\_sizes = [4.7, 8.5, 13.3, 15.9]

# Genetic algorithm parameters.

NUM\_INDIVIDUALS = 100 # Population size

GENERATIONS = 100 # Number of generations

MUTATION\_PROBABILITY = 0.1 # Probability of mutation

def calculate\_fitness(individual, dvd\_size):

# Total size of selected movies.

total\_size = sum(m["size"] for i, m in enumerate(movies) if individual[i] == 1)

# If total size exceeds DVD capacity, return a fitness score of 0.

if total\_size > dvd\_size:

return 0

# Check genre restrictions: no selection should include both restricted genres.

genres = {movies[i]["genre"] for i in range(len(movies)) if individual[i] == 1}

if genre\_restrictions[0] in genres and genre\_restrictions[1] in genres:

return 0

# Check pair restrictions: restricted pairs should not be selected together.

for restriction in movie\_restrictions:

if any(m["title"] == restriction[0] for i, m in enumerate(movies) if individual[i] == 1) and \

any(m["title"] == restriction[1] for i, m in enumerate(movies) if individual[i] == 1):

return 0

return total\_size # Return the total size if all restrictions are met.

def create\_individual():

# Create a random individual (a random binary selection of movies).

return [random.randint(0, 1) for \_ in movies]

def selection(population, dvd\_size):

# Calculate fitness for each individual and sort by fitness.

scores = [(ind, calculate\_fitness(ind, dvd\_size)) for ind in population]

scores = sorted(scores, key=lambda x: x[1], reverse=True)

# Select top 50% individuals for mating.

selected = [s[0] for s in scores[:NUM\_INDIVIDUALS // 2]]

return selected

def crossover(ind1, ind2):

# Single-point crossover between two parents.

point = random.randint(1, len(ind1) - 2)

return ind1[:point] + ind2[point:], ind2[:point] + ind1[point:]

def mutation(individual):

# Apply mutation with a given probability.

if random.random() < MUTATION\_PROBABILITY:

point = random.randint(0, len(individual) - 1)

individual[point] = 1 - individual[point] # Flip the binary bit at the mutation point.

def genetic\_algorithm(dvd\_size):

# Initialize population with random individuals.

population = [create\_individual() for \_ in range(NUM\_INDIVIDUALS)]

for \_ in range(GENERATIONS):

# Selection of the top individuals for reproduction.

selected = selection(population, dvd\_size)

new\_population = []

# Generate new individuals by crossover and mutation.

while len(new\_population) < NUM\_INDIVIDUALS:

parent1 = random.choice(selected)

parent2 = random.choice(selected)

child1, child2 = crossover(parent1, parent2)

mutation(child1)

mutation(child2)

new\_population.extend([child1, child2])

population = new\_population # Update population with the new generation.

# Find the best individual with the highest fitness value.

best\_individual = max(population, key=lambda ind: calculate\_fitness(ind, dvd\_size))

best\_fitness = calculate\_fitness(best\_individual, dvd\_size)

# Print the best solution for the given DVD size.

selected\_titles = [movies[i]["title"] for i in range(len(movies)) if best\_individual[i] == 1]

print(f"Best fit for DVD of {dvd\_size} GB:")

print(f"Selected movies: {selected\_titles}")

print(f"Used space: {best\_fitness:.3f} GB\n")

# Run the genetic algorithm for each DVD size.

for size in dvd\_sizes:

genetic\_algorithm(size)

## **6. Bibliografía**

<https://jarroba.com/algoritmos-geneticos-ejemplo/>

<https://www.youtube.com/watch?v=3Kzj2FNaua8>